SUPER CONTROL STATION
SF-7000

USER'S MANUAL

SEGA®

PREFACE

Supercontrol Staion SF-7000, containing a 3 1/2-inch compact floppy disk drive, 64K byte RAM, RS-232C terminal, and parallel printer terminal (applicable to Centronics), can be used as a high-performance peripheral device of SC-3000.

Beginners for SC-3000 BASIC should read this manual from the beginning, those who already know it, may ship the introductory sections. Before using the BASIC program, be sure to generate a backup copy of the disk BASIC. (See Chapter 1 for the procedure.)

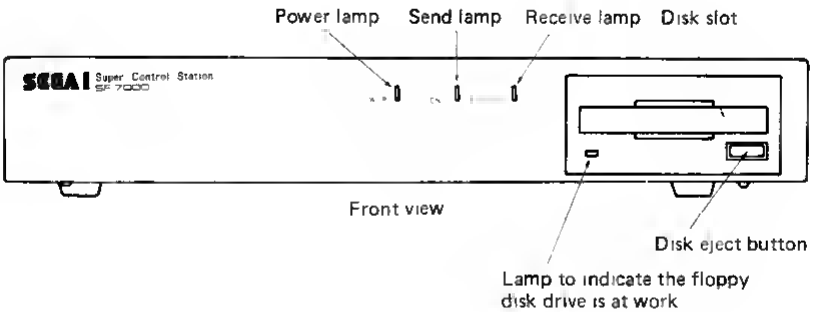
TABLE OF CONTENTS

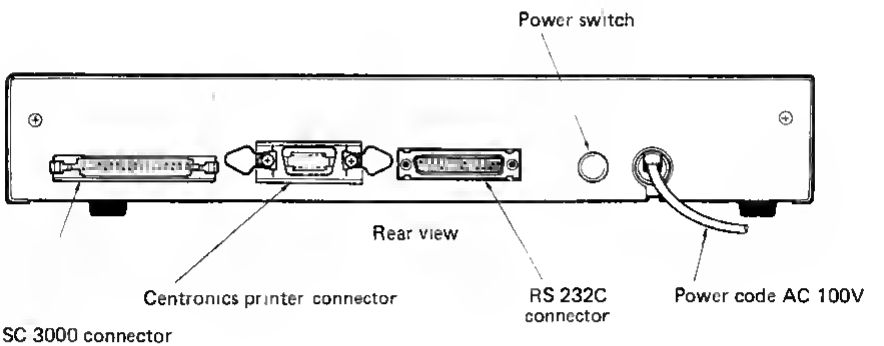
CHAPTER 7 SUPPLEMENT

CHAPTER 1 SUPER CONTROL STATION SF-7000

Parts Identification

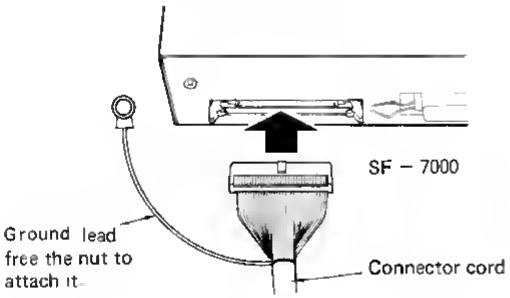
Unpack the SF-7000 and check the names of various parts of the SF-7000 using the figure below

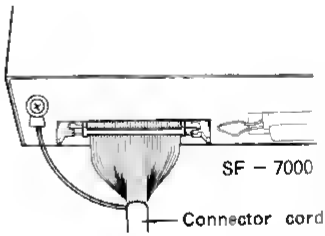


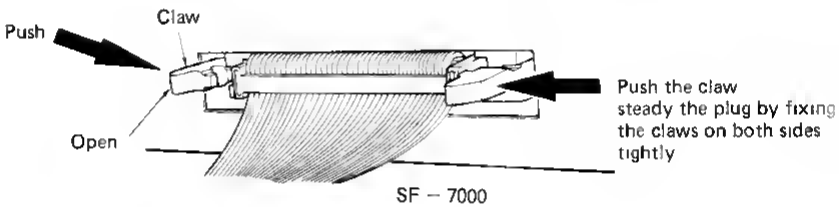


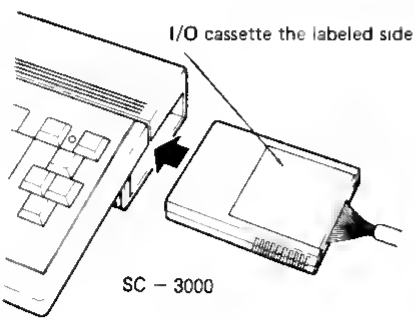
Connecting the SC-3000

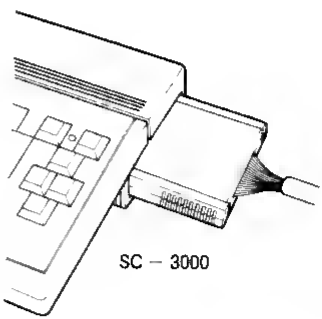
1 Connect the SC-3000 to the SF-7000 with the connector cord
Push the plug at one end of the connector cord deeply into the socket and steady it by pushing the claws attached on both sides of the socket inwardly (See the figure below).
(Push these claws open to release the plug.)
Then insert the I/O cassette at the other end of the cord into the slot in the SC–3000.



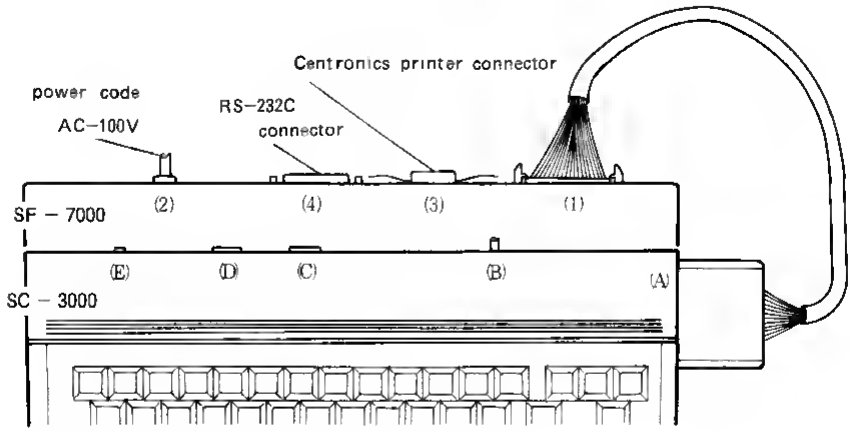








2 Connect various cords to the SC-3000.



SF-7000

(1) SF-7000 – SC-3000 connector cable

(2) Power code AC–100V

(3) Printer connector Centronics type (code optionally available)

(4) RS-232C connector (code optionally available)

SC-3000

(A) SF-7000 – SC-3000 connector cable

(B) Power socket DC–9V

(C) Printer connector SEGA SP-400 printer only

(D) Video socket, connected to the video input of the TV screen
(E) RF antenna socket, connected to the switch box

} Connect one of these according to your TV

3 Insert the power plugs of the SF-7000 and of the AC adaptor for the SC-3000 into the power socket.

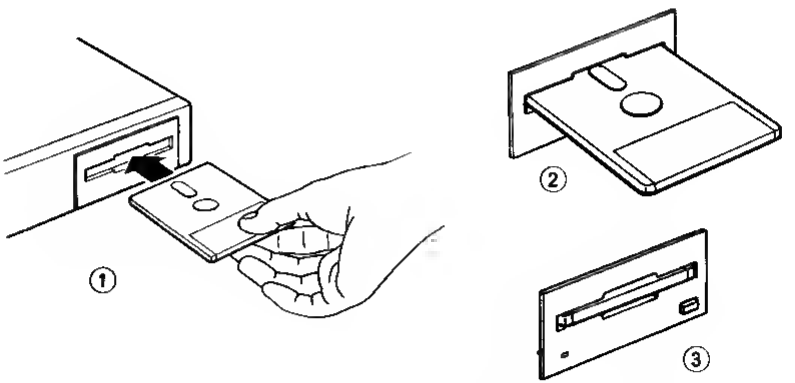
Now turn on the switches of the SF-7000 and the SC-3000.

> **CAUTION**
>
> **Turn on first the switch of the SF-7000, and then turn on the switch of the SC-3000**

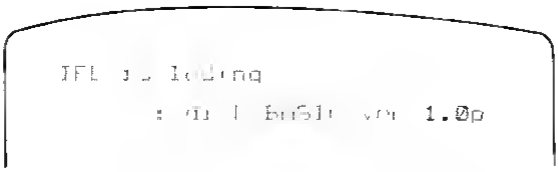
After the power on, the screen will remain dark for a few seconds, then after a screen-full of characters is flushed on the screen for less than about a second, the following message will appear on the screen:

```
disk not ready
 set disk and hit space key
```

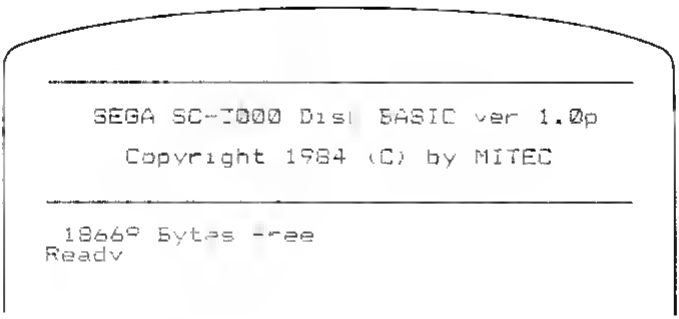
(This says that the disk drive has no disk in it, and that please insert your disk into the slot and hit the space key.)
At this time, insert the disk containing the Disk BASIC straight into the disk drive slot. The disk will drop somehow as it is set properly into the drive.



Now hit the space key to begin loading the Disk BASIC from the disk.



Do never touch the disk drive unit while the computer is displaying this message (the disk drive will start moving and the red lamp beside the slot will come alive)
When the transfer of the BASIC from the disk to the RAM in the SF-7000 is completed (the disk drive will stop and the lamp will go out), the BASIC interpreter will take hold of the screen and display the following message



The BASIC is now ready.
Before you start playing around with the BASIC, do take a backup copy of the disk lest you should destroy the disk and hence lose the BASIC forever.
To take a backup copy of your disk, refer to the article under the UTILITY program appearing afterward.
Do never remove disks while the disk drive is moving which is indicated by the red lamp beside the disk slot.
Disks are sometimes destroyed in this way.
Of course, you can remove the disk after the BASIC has come alive.
If the screen remains dark for more than 3 minutes, then power-off the devices and re-check the connections of each connector cables.

If it beeps successively while the screen remains dark, then consult the table below.

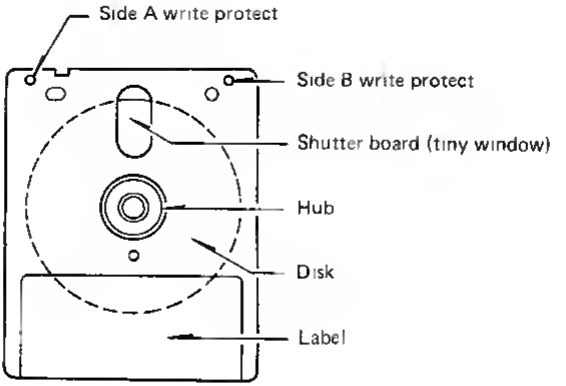
Possible causes

	Phenomena	Diagnostics
1	1 beep per second	IPL malfunction
2	2 beeps per second	RAM out of order
3	3 beeps per second	VRAM out of order

In any case verify connection of the connector cable.
Look inside the cartridge slot for the SC-3000 to see whether the pins are arranged in good forms. If any of the pins is crooked, troubles like above might well happen due to imperfect connection.
If you have found some pins crooked, do nothing by yourself and instead contact your retailer.
If you have found no abnormality in the connector cable, redo the setting from the first carefully following the previous instructions.
If the above mentioned steps all failed, then you must contact your dealer or retailer. Make sure the pins in the connector cable are arranged good forms.

Disk Handling

The SF 7000 uses the 3" compact floppy disk which is a thin circular disk made of magnetized vinyl enclosed in a hard plastic coverage to make it easy for manual handling



The tiny window opens when set into the SF-7000, and is closed while the disk is not used. Don't force it open

Since disks are made of magnetic substances, any sufficiently strong magnetic field will destroy the contents of disks. Do not place them near loud speakers or magnetic metals.

Using New Disks

New disks must be formatted before they can be used.

By formatting, we mean the marking of places on the circular disk from where data are written.

Any disk not formatted yet can not be used for program or data storage.

Formatting disks is simply called formatting.

Formatting is done in the UTILITY command.

The UTILITY Program

There are three commands to the UTILITY program.

F Format
For disk formatting

C Copy
For copying disks

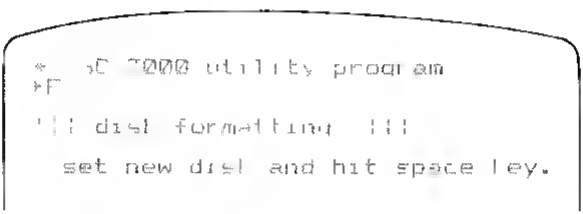
B Boot
For passing the control to the IPL

Copy uses a different format than usual SAVE to store data onto disks. While the BASIC system cannot be 'SAVE'd, it can be 'COPY'ed onto disks.
It is recommended that you take a backup copy of the BASIC system.

Formatting Disks

Type UTILITY followed by hitting the CR key to the BASIC interpreter.

Now hit the F key followed by CR.



Insert your new disk side A up into the disk drive unit and hit the space key. The red lamp will come alive and the formatting will begin.

After about 25 seconds the computer will display

formatting complete

which means the formatting has now completed
Now you can use the disk to write programs and data onto it But before you start writing data onto it, take the disk out from the slot and format the side B with the same procedure

Taking Backup Copies

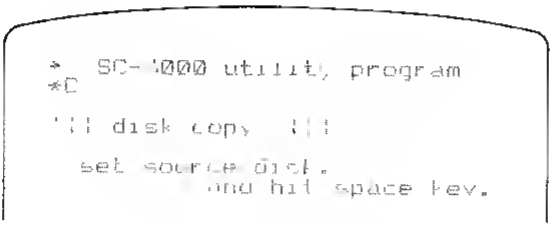
Taking Backup Copies of Floppy Disks

The surface of a floppy disk, although it is enclosed in a hard coverage is covered with magnetized substance and therefore the data it contains can easily be destroyed if it is placed within a strong magnetic field
Also you can easily destroy the content by handling it improperly Take backup copies of your important disks against such disaster

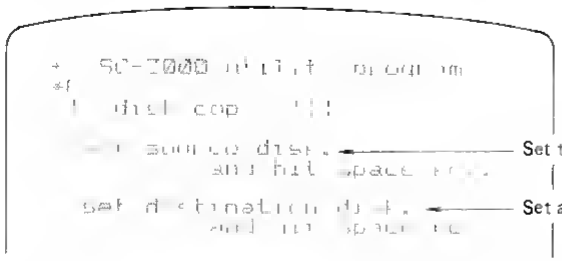
Method

The UTILITY program is available for this purpose
To take a backup copy, you need a source disk (the disk you need a backup copy of) and a destination disk (a newly formatted disk)
Now type TUILITY followed by CR

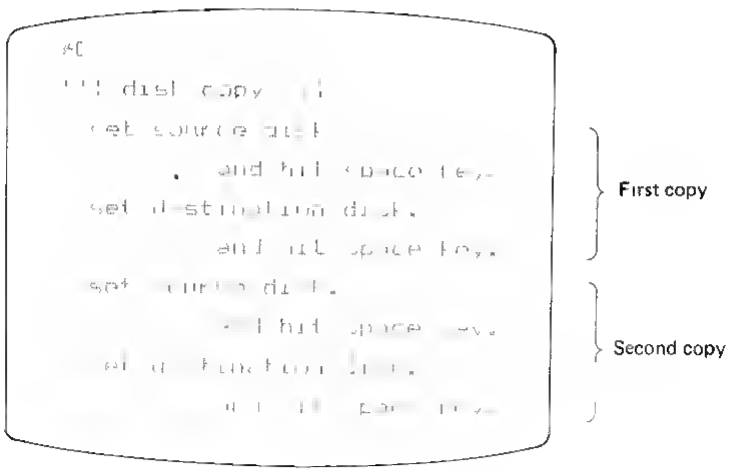
Then type C followed by CR



After the above message has been displayed on the screen insert the disk you want to have a copy of (source disk) into the drive and then hit the space key.
The lamp besides the slot will indicate that the copy is going on. The copy will end in a couple of seconds after having transferred the first ten trucks of data on the disk into memory and the following message will appear on the screen



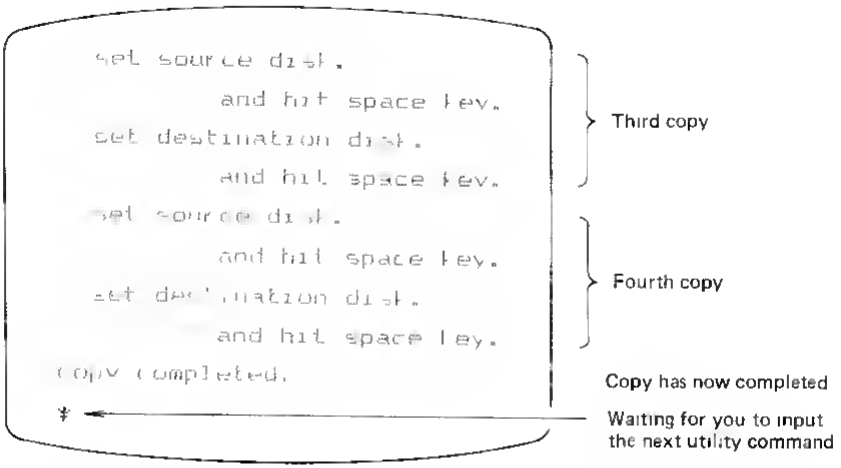
Now remove the source disk and insert a formatted, unused disk into the slot
Here note that if you used a disk with some programs already in it instead of an unused one, the programs in it would be erased and be replaced with the new programs now contained in the source disk
Now hit the space key. The lamp will come alive indicating the write has begun
The write will take some time and do never stop the disk unit while in this stage since otherwise the copy will be damaged
After about 45 seconds, the screen will look like this. The first quarter of your source disk has now been copied to the new disk. Exchange the disk with your source disk



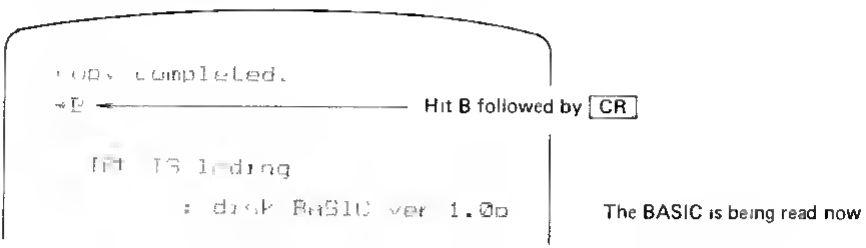
As you may have noticed ten trucks of data are copied one at a time. So you need repeat 4 times the above procedure to complete the copy of one side of a disk.
Upon completion of the copy, the message

copy complete

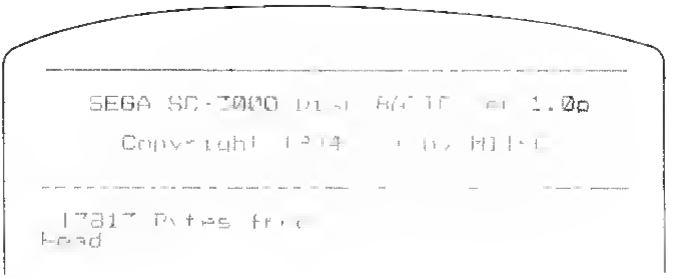
will appear on the screen
Don't leave the copy until the above message appears on the screen



Now don't remove the disk yet
Let's check whether the copy has correctly been performed Type B followed by CR in response to the '*' just under "copy completed."
The screen will look like the following and a boot is initiated to load the BASIC interpreter from the newly created disk



If the disk has been correctly copied, the title of the BASIC will appear on the screen



The message indicates the BASIC has been correctly saved
Put the source disk in a safe place and use the newly created disk instead of the original one
Now you can start programming with BASIC

The Role of Disk

Programs you develop with your computer are vanished the instant you power off the computer and it is such a tedious job to input all over again long programs to the computer
Fortunately there is a couple of devices other than the memory in the computer to store your precious programs so that you can use them from time to time without typing them all over again into the computer.
Those devices are

- cassette tapes (tape recorder)
- floppy disks
- buble memories

The SF-7000 has the 3" compact floppy disk drive installed in it and you can use the disk to save your programs as much as you need

The DISK BASIC (disk BASIC)

The DISK BASIC is an enhancement to the SC-3000 BASIC adding commands for disk control and for the RS-232C interface
The DISK BASIC, unlike the SC-3000 BASIC is loaded each time from disk into the RAM of the SF-7000 And once loaded the BASIC keeps running even though the disk is removed
Since the BASIC in RAM is erased on power-off, you use the SF-7000 for the first time

> As long as you use the BASIC on the SF-7000, you have no need of the BASIC cartridge. If you happen to possess it keep in a safe place in case you have to use it on the SC-3000 alone.

About the Connector

The Printer Connector

The SF 7000 has equipped with it a Centronics type printer connector. You can connect to it available Centronics printer with the following points in mind

The character codes for the printer must accord with the ASCII standard.
Though the ASCII standard is generally followed for English characters numeric characters and special symbols for typewriters, there are still some exceptions to this rule

- The printer interface must be the Centronics type
- The interface cable must be the one authorized by SEGA
  The pin arrangement must fit into the connector
- Some bit images cannot be printed-out with some printers
  Chose the right one to fit your purpose

The RS-232C Connector

This terminal is a communication port enabling you to communicate with other devices
The connector cable depends on the device you want to communicate
If, for example, you want to use an available acoustic-coupler, a cable for the coupler will be needed.
But from this terminal you can communicate with other personal computers.

To Use the Floppy Disk Properly

Read the following part carefully to properly handle the compact floppy disks

Disks used by the SF-7000

The disks that can be used with the SF-7000 are the SEGA floppy disk, or disks labeled as for single head drive
The disks for single head drive have the following mark on it.

Used with the single head floppy disk drive

○ The floppy disk drive

The SF-7000 uses a floppy disk drive (henceforce abbreviated to the drive) of the single head type hence disks for double heads cannot be used with the SF-7000. Please confirm the disk you are buying is a single-head type
Since both sides, A and B, of floppy disks (henceforth abbreviated to disks) are available for use in data storage, first use the side A and then turn it to utilize the other side B.

○ Setting the disk

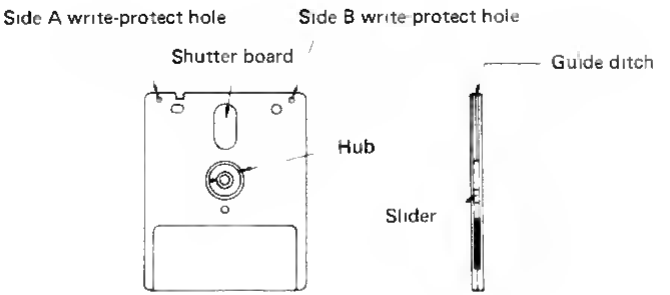
Insert the disk straight into the slot with the labeled part coming in last When the disk reaches to the bottom, the disk will sink in the unit and the eject button will come popping out

Removing the disk

Push the eject button A part of the disk will pop out of the slot

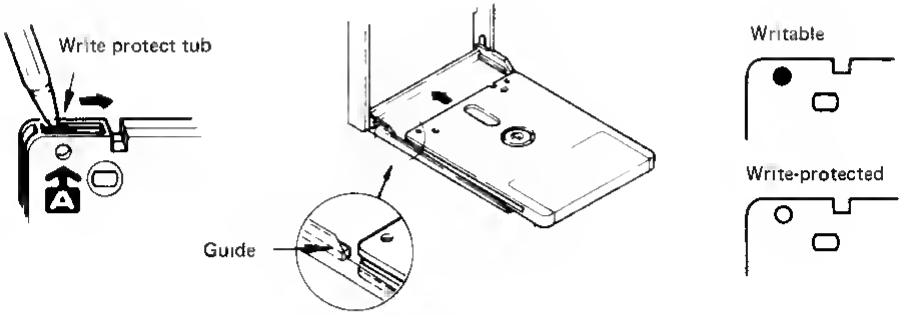
To use the compact floppy disk properly, remember the following

Parts Names



Proper Usage

- The compact floppy disk is for the single head drive. Turn it back to utilize the side B after you have used up the side A.
  Error will occur if you use this disk with double heads disk drive.
- To write-protect the disk, slide the write-protect tub with, for example, the tip of your ball point pen to open the write-protect hole completely (see the figure). In this way you can secure your important programs and data.
- To encase the disk, align the guide ditches of the case and of the disk as shown in the figure.



Proper Handling and Maintenance

1. Never force the shutter open and touch the magnetized surface inside.
2. Exercising excessive force either onto the shutter board or to the slider will damage the disk
3. Never deform the hub. Never use a disk with dust on it. Disks must be kept free of dust.
4. Keep the disk out of thinner, alcohole or phleon
5. Don't use rubber erasers against the disk
6. Don't eat or smoke while there are disks around you
7. Encase the disks while not in use.
8. Evade high temperature, excessive humidity or direct sunshine.
9. Never leave the disks in a magnetic field.
10. Keep them out of dust

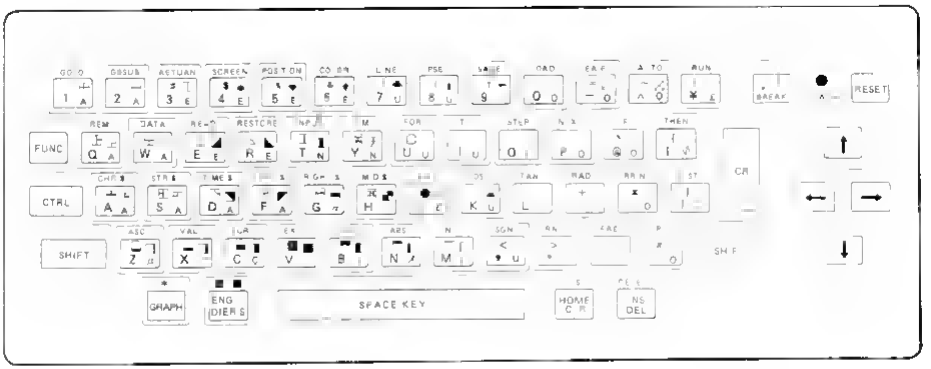
Working and Preservance Environment

	Temperature	Humidity
Working environment	10 – 51.5°C	20 – 88%RH*
Preservance environment	4 – 51.5°C	8 – 80%RH
Shipping environment	-40 – 51.5°C	8 – 90%RH

* Maximum wet bulb temperature below or equal to 29°C

CHAPTER 2 HOW TO USE BASIC

Using the Keyboard



The KEYBOARD has keys on which letters, numbers and symbols are written Key layout and spacing are the same as you would find on a typewriter, so you can press the keys with your fingers easily

To get a clear screen so that you can experiment with typing, press the key marked

HOME
CLR

HOME CLEAR

Press the keys 1 and then 2 which should show 12

Now you have 12 displayed on the screen If you press keys by themselves, the lower characters or symbols written on the keys will be displayed on the screen

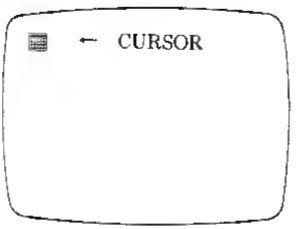
When SHIFT key is held down and then the 1 key is pressed ! appears on the screen

You will notice that there is a small square blinking on and off When you type something, it replaces the square, and the square moves one place to the right—where the next character to be typed will appear.

This is called the CURSOR

To move the cursor around the screen use the keys marked ← → ↑ ↓ By using these keys you can move the cursor without erasing the characters

When you move the cursor directly on top of a character you have already typed, you can then type a new character to replace the old one. This is very important when you are correcting any mistakes you might have made in typing your new programs



The type of the CURSOR varies depending on respective modes.

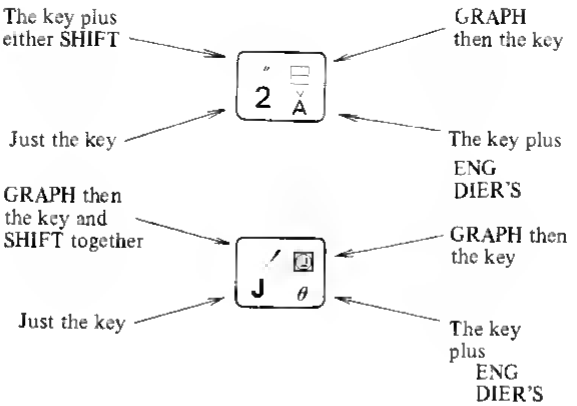
Alphanumeric mode
Dieresis mode
Graphic mode *

That is when you are typing the graphics characters you will have the cursor looking like a star *. All other characters will have the square.

To display the graphics characters press the GRAPH key which is in the lower left hand side of the keyboard.

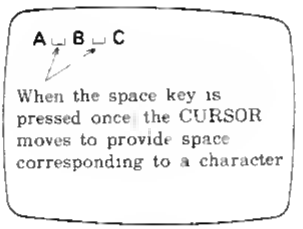
The CURSOR will change. To return the CURSOR to the original position from graphic mode, press the GRAPH key again. While the cursor is a star (*) only the graphics characters will be displayed

The following diagrams explain what you have to press to display the various characters. When more than one key has to be pressed they must be pressed together. (Except for the GRAPH key which must be pressed first.)



If you want the small letter (j), then push the key and SHIFT (The opposite to what you would do on a typewriter)

To type a space between characters and symbols use the SPACE key. This is the elongated key on the bottom row.



If you type over another character with this key it will replace it with a space. In computer programming, space is handled as information, in the same way as characters

By using the FUNC key (Function) with other designated keys you can enter many of the common BASIC words with a single keystroke.

For example

By identifying this key with GOTO written on the upper part of it, you have one of the command statements used in BASIC

While holding down the FUNC key, press the other keys and various command statements will be displayed This is very useful when typing programs

SPECIAL KEYS

Now you notice that the screen is full of characters and symbols which were previously entered by keys

To remind yourself which keys are which, we recommend you affix the silver adhesive labels supplied with your computer pack, to the appropriate keys as indicated

It's no use leaving unwanted characters and symbols on the screen. We can clear the whole screen, using the following key

HOME/CLR (Home/Clear)

When this key is pressed, characters on the screen are erased and the CURSOR returns to the upper left "Home" position. Use this key whenever you want to clear the screen.

When HOME/CLR key is pressed while holding down the SHIFT key, the screen will remain uncleared but the CURSOR returns to the home position

CR (Carriage Return) or (Return)

It is important to remember that as you type characters on to the screen, they will not be stored inside the computer until the CR key is pressed

Type any character and press the CR key You notice "Syntax Error" displayed on the screen This is because instructions for computers need to be written a certain way, i.e. in the BASIC language. This is called computer Syntax, and if this Syntax is wrong, errors will occur.

Whenever these errors occur, you should refer to the error message table in the appendix for further information.

INS/DEL Insert/Delete

The INS/DEL key is used when deleting or adding characters one by one.

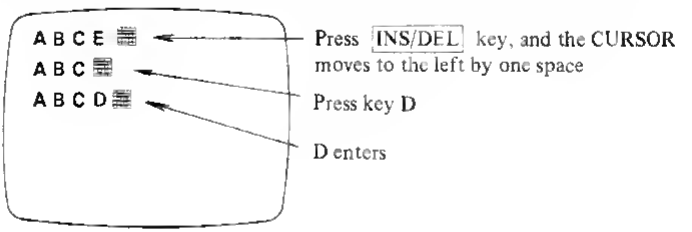
INS (Insert) refers to the addition of characters.

DEL (Delete) refers to the deletion of characters.

Type [A] [B] [C] [E].

If you want to change this to

[A] [B] [C] [D] the CURSOR will move backward by one character when the [INS/DEL] key is pressed, and then character E is erased. Now, press D and the correction has been made to A B C D

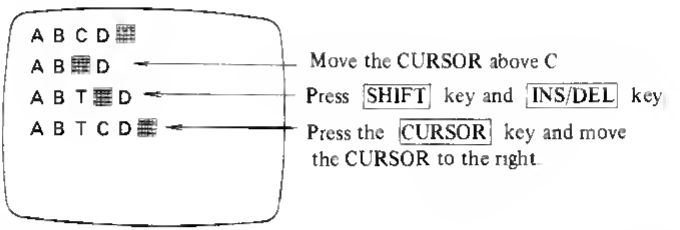


Now let's put a character in between the B and C of A B C D

Bring the CURSOR over the C of A B C D

While holding down the [SHIFT] key, press the [INS/DEL] key. You notice that the CURSOR blinks quicker than before. Input any character.

The character which was entered just now is positioned after B, and C D moves to the right by one character space



When the insert mode is used, as many characters as required can be entered, while the CURSOR is blinking quickly.

To get out of the insert mode and back to the original state.

1. Press [CR] key or,
2. Press CURSOR control key, (one of the light grey keys with arrows) or,
3. Press [SHIFT] + [INS/DEL] key.

By pressing either one of the above keys, the normal condition is restored. When the program is corrected, the MEMORY cannot be rewritten unless the CR key is pressed

Try all this out by typing in a line of letters and moving the CURSOR to the middle, pressing SHIFT + INS/DEL then typing more characters. Try pressing INS/DEL without the SHIFT key and see the result

[BREAK]

The [BREAK] (Screen shift/break) key when used on its own, is used for stopping programs during program run

The screen will also change when the [BREAK] key is pressed while holding down the [SHIFT] key. By pressing these keys again the screen will change back

This is used for changing the screen, as the computer has two screens. One screen is the text screen for entering programs, and the other is the graphic screen for displaying graphics

Another way of stopping a program is to use the [RESET] (Reset) key

[RESET] (Reset)

If you press the [RESET] key during a program run, or when problems appear on the screen, the screen returns to what it was when the power was turned on. This will occur within one or two seconds after you press the key

When you press this key, the computer stops processing and the size of the MEMORY which has not been used, is displayed in the format

```
xxx Bytes Free
```

Even if the key is pressed, programs which were entered will remain in the memory

Control codes

Key operation	PRINT CHR $ (Value)	Function
CTRL + A	PRINT CHR $ (1)	NULL No character
C	–	BREAK Stops program execution
E	5	Clears characters after the cursor
G	7	BELL Makes beep sound
H	8	DEL Deletes a character
I	9	HT Horizontal tab
J	10	LF Line feed
K	11	HM Returns the cursor to the home position
L	12	CLR Clears the screen
M	13	CR Carriage return
N	14	Keyboard shift (kana ↔ alphanumeric)
O	15	Screen shift (text screen ↔ graphic screen)
P	16	Standard character size
Q	17	Character size doubled horizontally (SCREEN 2)
R	18	INS Insert
S	19	Key entry for capital letters (A Z), no shift
T	20	Key entry for small letters (a-z), no shift
U	21	Clears the current line and returns the cursor to the left margin
V	22	Normal mode
W	23	GRAPH Shift (key entry graph mode ↔ letter mode)
X	24	Click sound setting (on ↔ off)
Z	26	Printer selection (#1 ↔ #2)
–	28	⇨ Cursor movement
–	29	⇦ Cursor movement
–	30	⇧ Cursor movement
–	31	⇩ Cursor movement

To specify a control code in the program, enter the associated PRINT CHR$ (value).

1. Key entry

When BASIC is started, a title screen is displayed and BASIC becomes ready to use. The digits displayed on the left side below the title indicate the number of memory bytes available.

1.1 Operational Modes

BASIC has two operational modes

Direct mode

A command or statement entered without a line number can be executed only by pressing the CR key

Enter the following

To get the quotation mark " press this key while holding down a shift key.

HOME/CLR P R I N T " B A S I C SPACE T E S T " CR

```
10 PRINT "BASIC TEST"
RUN
BASIC TEST
Ready.
```

A syntax error occurs unless a PRINT statement is specified at the beginning

○ Indirect mode

If a command or statement entered is assigned a line number, it is not executed only by pressing the CR key. This method is called "program input". Assigning a line number implies storing the associated command or statement in the memory as a program. The CR key must be pressed to store the line in the memory. If it is not pressed, the line is not stored in the memory or it is stored together with the next line, causing an error.

Example

Enter 1 0 P R I N T " B A S I C T E S T " CR

Enter R U N CR RUN is a command that executes the program

```
BASIC TEST
Ready
```

A program consists of procedure lines that specify computer jobs. Assign numbers to the lines in the oder of execution. Use a RUN command to execute the program.

1.2 Statement

A BASIC program consists of sentences.
A sentence contains commands, statements, functions, etc. to specify the procedures to be performed by BASIC.
Two or more statement separated by a colon (:) may be written for one line number. This is called a multistatement. A multistatement having one line number can contain up to 255 characters.

Statement example 1

```
10 FOR N=0 TO 10
20 PRINT N
30 NEXT N
```

} Program consisting of three lines

Statement example 2

```
10 FOR N=0 TO 10:PRINT N:NEXT N
```

This is a multistatement

Also in the direct mode, programs can be written in multistatements.

1.3 Line Number

Each BASIC program line must begin with a line number which is an integer ranging from 1 to 65535. Program lines are stored in the memory and executed in the order of line numbers. Line numbers are also used as markings to indicate program flow and editing. Usually, it is better to assign line numbers stepped by 10 as 10, 20, 30, . . . Later, additional lines can be assigned line numbers 5, 11, 25, . . .
Lines may not be entered in the order of line numbers, they are automatically rearranged in te the ascending order when stored in the memory.

1.4 Usable Characters and Symbols

BASIC programs can use capital letters, small letters, digits, katakanas, special symbols, and graphic characters. They are assigned code numbers. (sSee the character codes in the Appendix.)
Programs may also contain control codes.

1.5 Special Symbols

Some symbols used in BASIC programs have special meanings.

1 Double quotation mark (")
Characters enclosed in a pair of double quotation marks ("), for example, "ABC", are treated as a character string. When digits are enclosed in quotation marks, for example, "123", they are also treated as a character string, not numerals, and cannot be used for calculation.

Example: `"ABC"`

2 Number symbol (#)
Assign this symbol to a disk file command

Example: `INPUT#`

3 Dollar symbol ($)
Assign this symbol to a character string variable to recognize it from a numeric variable

Example: `A$="BASIC"`

4 Ampersand (&)
Use this symbol together with H (for example, $H3C00) to indicate a hexadecimal number.

Example: `&HF`

5 Minus sign or hyphen (-)
Use this symbol as a minus sign, to specify subtraction, or to specify the range of lines in a LIST or DELETE statement. It may also be used in a LINE statement

Examples: `PRINT 8-6` or `LIST 50-60`

6 Semicolon (;)
Use a semicolon as a delimiter in a PRINT statement. When this symbol is used, the value is displayed on the right side of the first expression.

Example: `PRINT "A=";A`

7 Colon (:)
Use a colon as a delimiter in a multistatement.

Example: `C=A+B:PRINT C`

8 Comma (,)
Use a comma to delimit parameters listed in a PRINT, INPUT, or other statement.

Examples:
```
INPUT A,B,C
PRINT A,B,C
```

When a comma is used in a PRINT statement, the second value is displayed at the 20th column from the screen left margin.

9 Interrogation mark (?)
This mark can be used in place of a PRINT statement

Example: `?"BASIC"`

10 Space
Spaces in programs have no special meanings. Spaces entered between characters in a statement have no influence to the execution result
Spaces in a character string are treated as characters.

Example: `PRINT "   BASIC   "`

In the example above, the character string consists of 10 characters including spaces

2. Variables

2.1 Variable types and operations

A variable consists of one alphabetic character or two characters and it can be assigned a numeric value of up to 11 digits or a character string of up to 255 characters.
An alphabetic character and a digit may be used in combination in a variable, but it must begin with an alphabetic character. Since a variable may contain up to two characters, only one digit can be used. If a variable name (characters specified in the variable) consists of more than two characters, the first two characters are used and the third and subsequent characters are ignored. Command, statement, and function names cannot be used as variables.

Variable names

```
BASIC=1000
BASE=248
```

} In both cases, the variable name is BA.

Variable types

- Variable
  - Numeric variables — A, AB, C1
  - Character variables — A$, NA$, D5$
  - Array variables
    - Numeric array variables — E(1), G(2, 2)
    - Character array variables — D$(1), H$(3, 3)

A character variable and character array variable must be assigned a dollar sign ($) to recognize them from a numeric variable.
If the variable type is illegally used, error message "Type mismatch error" is displayed.
When entering a character variable with an INPUT statement, digits are treated as characters. Digits assigned to a variable as a character string do not have a numeric value and they cannot be used for arithmetic operations.
Even if a variable and an array have the same name, they are used as different variables.

2.2 How to use variables

Numeric values and character strings used by programs are called data. Data is assigned to variables, then stored in the memory. Variables have names consisting of symbols.

```
A=153278
```

In the example above, a numeric value 153278 is assigned to variable A. After this numeric value is assigned, variable A can be used as 153278 in an expression. If another numeric value is assigned to A in the same program, A has they new value. Since the contents can be changed, A is called a variable.

```
10 A$="A[illegible]"                ← Specifies a character variable.
20 INPUT "A=";A                     ← Specifies assigning a numeric value to variable A
30 INPUT "B=";B                     ← Specifies assigning a numeric value to variable B.
40 C=A+B                            ← Specifies assigning the sum of A and B to variable C.
50 PRINT A$;C                       ← Specifies displaying the values of character string
60 GOTO 10                            A$ and variable C
RUN
A=35
B=50
A+B= 85
A=                                  ← Specifies returning to line number 20 and waiting
Break in 20                           for entry of variable A.
```

When this program is executed, the program displays A=, then waits for an entry of a numeric value. After entering a numeric value and pressing the CR key, the program displays B=. If another numeric value is entered and the CR key is pressed, is displayed, where indicates the sum of the numeric values entered. After line 60, the program returns to line 20 and waits for an entry to A.

The value of variable C changes when new values are entered to variables A and B.

Character variable A$ specified in line 10 is used in line 50. The variable value stored in the memory is the same as that entered last. Check it as follows.

Enter ? A CR (do not specify the line number). Check B and C in the same way.

Note

The equal sign (=) used in a variable expression differs from that used in an arithmetic expression, it means assigning the expression on the right side to that on the left side.

The basic format is as follows.

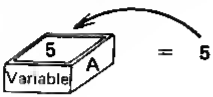
LET A=20

LET can be omitted; actually, A=20 brings the correct result.

2.3 Variables illustrated

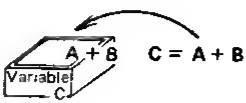
The following illustration will help you to understand variables. Assume that a box is a variable. This box can contain a numeric value or characters.

LET A=5 This expression assigns 5 to A.



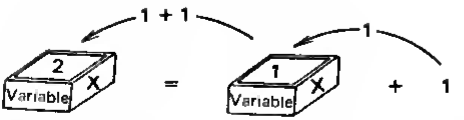
Input 5 in box A.

LET C = A + B



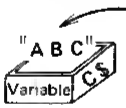
Input the sum of A and B in box C.

X = X+1 LET is omitted.



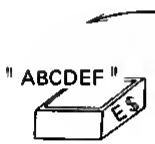
The equals symbol (=) in X = X + 1 does not mean equality It assigns the result of the expression on the right side to that on the left side

This expression is not mathematically correct, but it is often used for accumulation in computer programming.
Character string variables can also be used like numeric variables.



="ABC" C$="ABC" Input character string ABC in box C$

="DEF" D$="DEF" Input character string DEF in box D$



="ABC"+"DEF" E$=C$+D$

Input the concatenation of ABC and DEF in box E$

2.4 Numeric value range

The computer internally processes numeric values of up to 12 digits and it displays those of up to 11 digits. The floating decimal point system is used.

Numeric value range

± 9.999999999999E−99

≀

± 9.999999999999E+99

If a numeric value consists of more than 11 digits or it is less than 0.01, it is processed in the exponential form

1 2 3 4 5 6 7 8 9 1 2 5 → 1.2345678913E+11

Exponential form

The above expression means 1 2 3 4 5 6 7 8 9 1 3 x 10^{11}

0 . 0 0 1 2 3 → 1 . 2 3 E − 0 3 → This is 1 23 x 10^{-3}

Exponential form

3. Constants

3.1 Numeric constants

- Integer form : 3 、 5 、 8 5 7 3 2 4
- Decimal point form : 0 . 2 、 . 5 、 6 7 . 3
- Exponential form : 3 E 9 9 、 6 E − 3 、 1 . 3 E + 7
- Hexadecimal form : &H64 (100 in decimal), &HF (15 in decimal)

3.2 Character constants

Enclose characters and symbols in a pair of double quotation marks (").

" ABCdef "	Characters ABCdef
" "	Character NULL (length 0)
" " " "	Symbol "
" A3 " " 64 "	Characters A3 " 64

Also enclose symbols having special meanings in the program in double quotation marks. (Example: ",")

4. Operations

4.1 Operators

The computer can perform arithmetic and logical operations. Arithmetic operations use arithmetic operators, some of them are special symbols

Plus (+)	Addition
Minus (-)	Subtraction
Asterisk (*)	Multiplication
Slash (/)	Division
(^)	Exponent

An arithmetic expression is evaluated from the leftmost item; if it contains parentheses, the items enclosed in parentheses are evaluated first. Pairs of parentheses can be nested, but braces and brackets cannot be used. Items in a pair of parentheses are evaluated in the ordinary order. Since an arithmetic expression is written in a line, fractions and power items are written in a different form.

$3A+B$	3*A+B
$\frac{A}{B}+5$	A/B+5
$\frac{A+B}{C}$	(A+B)/C
X^2+2X+1	X^2+2*X+1
A^{x^2}	A^(X^2)
$(A^x)^2$	A^X^2
$A(-B)$	A*(-B)

Calculation error conditions

1. An attempt is made to divide by 0. The divisor must not be 0.
2. An expression is written in too many columns (overflow).

4.2 Relational operators

A relational operator is used to compare two numeric values. If the comparison result is true, -1 is returned; if it is false, 0 is returned. Relational operators are used to change the program flow in an IF statement.

Relational operators

=	The left side is equal to the right side.	X=Y
<>	The left side is not equal to the right side.	X<>Y
<	The left side is less than the right side.	X<Y
>	The left side is greater than the right side.	X>Y
<=, =<	The left side is less than or equal to the right side.	X<=Y, X=<Y
>=, =>	The left side is greater than or equal to the right side.	X>=Y, X=>Y

An equals sign (=) used in an assignment statement means assigning, not equality. A=0 and X=X+1 not in an IF statement are assignment statements.

How to use relational operators.

```
IF X=0 THEN 500 ← This is equivalent to GOTO500
IF X=>37 THEN X=0:Y=Y+1
IF A<>B THEN 200
```

4.3 Logical operators

Logical operators can be used to check more than one condition.

1. NOT (negation)

X	NOT X
1	0
0	1

2. AND (logical product)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

3. OR (logical sum)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

4. XOR (exclusive logical sum)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

4.4 Functions

A function specifies a certain opeation for te given argument(s) and returns a certain value as the result.

Disk BASIC has intrinsic functions such as numeric functions and character string functions. SIN (sine) and LOG (logarithm) are numeric functions and CHR$ and TIME$ are character string functions. In addition, there is a user function DEFFN.

These functions are explained later.

4.5 Character string operation

Addition, comparison operations, and logical operatios can be performed on character strings. Since characters are compared by their character code values, they can be sorted.

4.6 Calculation priorities

Calculation is performed in the order of the following priorities.

1 Expressions enclosed in parentheses
2 Functions
3 Exponent (power)
4 Minus sign (-)
5 *, /
6 +, -
7 Relational operators
8 NOT
9 AND
10 OR

	Symbol	Where usable		Description	Priority
		Numeric variable	String variable		
Arithmetic operation	∧	○	×	Power (0∧0=1)	1
	+	○	×	Code +	2
	−	○	×	Coad −	
	*	○	×	Multiplication	3
	/	○	×	Division	
	MOD	○	×	Residual	4
	+	○	○	Addition (Character combination in case of strig variables)	5
	−	○	×	Subtraction	
Relative operation	=	○	○	Equal to (−1 for true, 0 for false)	6
	<>	○	○	Not equal to (−1 for true, 0 for false)	
	>	○	○	Greater than (−1 for true 0 for false)	
	<	○	○	Less than (−1 for true 0 for false)	
	>=	○	○	Greater than or equal to (−1 for true, 0 for false)	
	<=	○	○	Less than or equal to (−1 for true 0 for false)	
Logical operation	NOT	○	×	Logical denial	7
	AND	○	×	Logical product	8
	OR	○	×	Logical sum	9
	XOR	○	×	Exclusive OR	10

4.7 Calculation precautions

The disk BASIC provides internal decimal opeations on 12 digits and it displays the result in 11 digits The calculation error is extremely reduced

For exponential opeations, multiplications are not used and the results may be indicated by approximations They contain little errors, but they are corrected for better readability.

The result of 3 4 is indicated as 80.999999999

Since the correct result of the above expression is 3x3x3x3=81, it should be expressed as follows.

INT((3 4)+0.1)

When the fraction value is significant, the result has little error

The Disk BASIC calculates exponention as follows:

x^y

1. $y = 0$ $x^y = 1$ $(0^0$
2. $x = 0$ $x^y = 0$
3. $x < 0$ Uncalculatable
4. $x < 0$ $x^y = e^{y \cdot \log_e x}$

CHAPTER 3 DISPLAY SCREEN

1. Screen

SC-3000 provides two independent screens which cannot be used simultaneously. Select the proper screen according to the operation.

Screen 1 Text screen

BASIC initially displays the text screen. Characters can be directly input to this screen. This screen cannot display the execution results of graphic statements such as LINE and CIRCLE statements.

Screen 2 Graphic screen

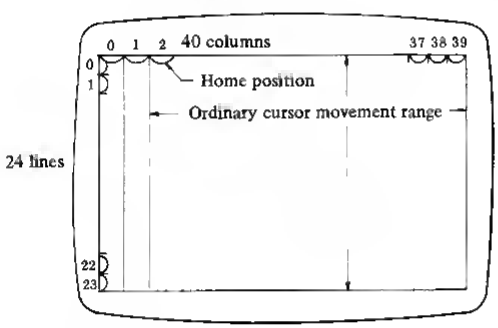
Usually, a test screen is displayed and a graphic screen is not visible. Specify a SCREEN statement to display the graphic screen. This screen displays graphics specified with graphic statements. After execution on the graphic screen is over, the program automatically displays the text screen. To continuously display the graphic screen, set an endless loop in the last line.

2.1 Text screen configuration

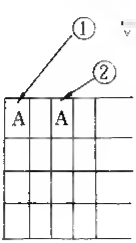
A text screen has a configuration of 38 columns x 24 lines. Use of the two leftmost columns on the screen are restricted, because some CRTs do not clearly display this area. Characters can be displayed in this area with a VPOKE statement. The address of the leftmost column on the screen is as follows:

&H3C00 (hexadecimal)
15360 (decimal)

Display screen



2.2 Coordinates on text screen



Cursor origin ($x = 0$, $y = 0$)

Coordinates range of CURSOR statement $x = 0 \sim 37$

$y = 0 \sim 23$

The screen origin 1 is at the second column leftward from the BASIC home position 2.
The BASIC home position implies the cursor origin; its coordinates to be specified in a CURSOR statement are x=0 and y=0.
The actual screen origin is at the second column leftward from the home position; its VRAM address is &H3C00 (15360 in decimal notation).
Execute the following statements to check the above.

① [illegible] or [illegible]

&H41 (65 in decimal notation) is the character code of letter A.

② [illegible]

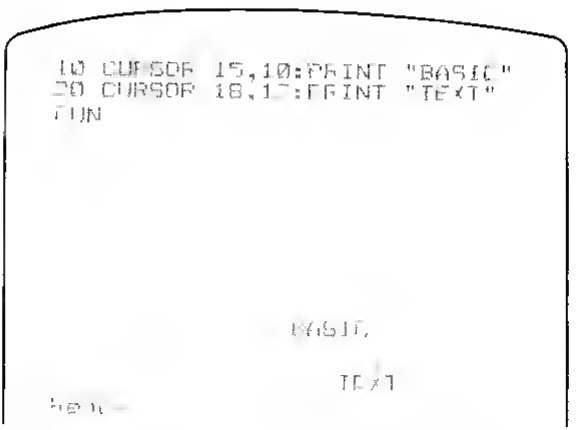
Usually, BASIC uses the home position as its origin. To display characters on the full screen, use a VPOKE statement.
When a PRINT statement is executed on a text screen, characters are displayed from the screen top. To display characters at the desired positions, move the cursor with a CURSOR statement. Characters are displayed from the position specified in the CURSOR statement. Determine the position according to the number of characters.

The address calculations on the text screen are carried out as follows.

$$\text{Address (text)} = y * 40 + x + \&H3C00$$
$$\text{where } (x = 0 \sim 39,\ y = 0 \sim 23)$$

For the data to be sent, the ASCII code of the corresponding character is applicable (0 ~ 255 in decimal numerals and 0 ~ &HFF in hexadecimal numerals).

Example 1.



To specify the position in a CURSOR statement, enter the coordinates in the order of x- and y-axes.

Example 2

Variables can be used to specify coordinates in a CURSOR statement. This method is useful when changing the display position.

Example 3

```
10 FOR Y=5 TO 15
20 CURSOR X,Y:PRINT "I"
30 NEXT Y
```

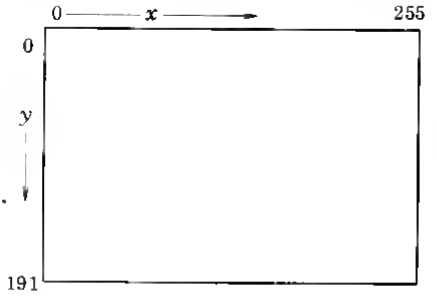
Specify a FOR statement to change only the y-coordinate

3.1 Graphic screen configuration

A graphic screen configuration is 256 (horizontal) x 192 (vertical) picture elements. Usually, a picture element is treated as a dot. In graphic screen explanations, both picture elements and dots are used, but they have the same meaning, except for special operations.
The origin of a graphic screen is at the left-top corner, like for a text screen, but some CRTs may not clearly display the origin.

3.2 Coordinates on graphic screen

The graphic screen is managed in the units of dots (picture elements). In LINE, PSET, and CURSOR statements, specify the coordinates in the following range.

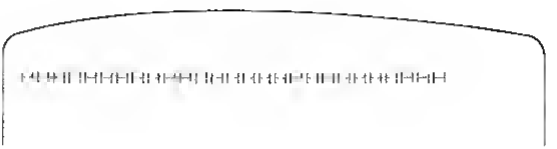


Effective range of coordinates
x-axis: 0 ~ 255
y-axis: 0 ~ 191

When displaying characters with a CURSOR statement, the coordinates must be specified so that the characters are not overlapped on the screen. Since a character consists of 6 (horizontal) x 8 (vertical) dots, more then 6 horizontal dots and more then 8 vertical dots must be reserved betwene the specified coordinates.

```
10 SCREEN 2,1:CLS
20 Y=16
30 FOR X=0 TO 100 STEP 6
40 CURSOR X,Y:PRINT "A";
50 NEXT X
60 GOTO 60
RUN
Break in 60
```

Graphics screen displayed



After characters or graphics are displayed on a graphics screen, this screen is automatically changed to a text screen. To keep the graphics screen display, specify an endless loop in line 60. Press the BREAK key to terminate program execution.

3.3 Drawing figures

Draw a line on the screen.

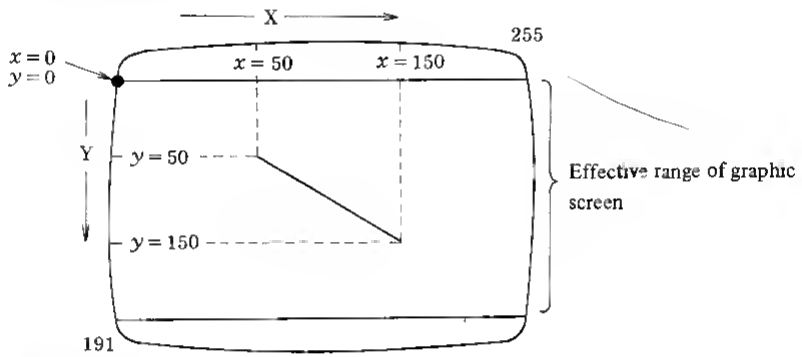
Before drawing a figure on the screen with a LINE or CIRCLE statement, specify the graphic screen as follows:

```
SCREEN 2,2:CLS
```

Write this statement for screen selection at the beginning of the program. The program execution result can be seen on the graphic screen.

To draw a line, specify the coordinates of the line starting point and those of the line ending point in a LINE statement. The specified point must be a crossing of the x- and y-axes.

```
10 SCREEN 2,2:CLS
20 LINE(50,50)-(150,150),6  ← Color number
30 GOTO 30                    (See the COLOR statement.)
RUN
Break in 30
```



For a different line, change the coordinates.

After changing the coordinates or characters with the CURSOR key, press the CR key.

B specifications: Drawing a box

```
LINE(50,50)-(150,150),5,B
```

A LINE statement can draw a box (Zrectangular), in addition to a line. To draw a box, specify B (box). When B is specified in a LINE statement, it draws a box whose diagonal line is specified in the same LINE statement.

F specification

```
LINE(50,50)-(150,150),5,BF
```

This LINE statement paints inside the box with the specified color.

Drawing a circle

To draw a circle, specify the center coordinates

```
10 SCREEN 2,2:CLS
20 FOR R=10 TO 70 STEP 5
30 CIRCLE (120,90),R,5,1,0,1
40 NEXT R
50 GOTO 50
RUN
Break in 50
```

See the explanation on the CIRCLE statement for details

The following statements and commands are valid only on the graphic screen (They are invalid on tn text screen.)

SCREEN	POSITION
LINE	BLINE
CIRCLE	BCIRCLE
PSET	PRESET
SPRITE	MAG

4. Addresses on graphic screen

The PVOKE address at the leftmost end of the screen in &H0000. Data specified at this address is displayed on the screen. See the explanation on a PATTERN statement for data.

Example

```
VPOKE &H0011,&HFF
Ready

VPOKE &H0017,255
Ready
```

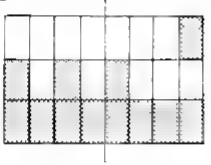
The graphic screen displayed directly with commands can be seen only for an instance. To see the screen again, press the BREAK key while pressing the SHIFT key. The screen is scrolled down. When these keys are pressed again, the text screen is displayed.

```
10 SCREEN 2,2:CLS
20 FOR V=0 TO 7
30 READ D$
40 VPOKE &H0010+v,VAL("&H"+D$)
50 NEXT V
60 DATA 01,03,07,0F,1F,3F,7F,FF
70 CV=&H2000
80 FOR V=0 TO 7
90 READ D$
100 VPOKE &H0010+V+CV,VAL("&H"+D$)
110 NEXT V
120 DATA 5F,5F,5F,5F,8F,8F,8F,8F
130 GOTO 130
RUN
Break in 130
```

Specify data at addresses &H0010 to &H0017 to draw a triangle. The color table addresses begin at &H2000. Specify the color with by color number

Addresses on graphic screen

Addresses on the graphic screen begins at VRAM address &H0000. An address stores 8-bit data, divided to four high-order bits and four low-order bits. The high- and low-order bits are indicated in the binary notation and they are displayed in the hexadecimal notation. Thus, two hexadecimal digits can be used to write the contents of an address



High-order bits | Low-order bits	Binary notation	Hexadecimal notation	Decimal notation
0 0 0 0 | 0 0 0 1	0 1	0 1	0 1
1 0 1 0 | 1 0 0 0	A 8	1 6 8	
1 1 1 1 | 1 1 1 1	F F	2 5 5	

The computer can handle both decimal and hexadecimal numbers. &H must be assigned to hexadecimal numbers. Decimal 10 is equivalent to hexadecimal &HA.

In VRAM addressing, eight horizontal bits (one byte) has one address. The bytes in the leftmost column are assigned addresses &H0000 to &H0007 from the top. In the same way, the bytes in the next column are assigned addresses &H0008 to &H000F.

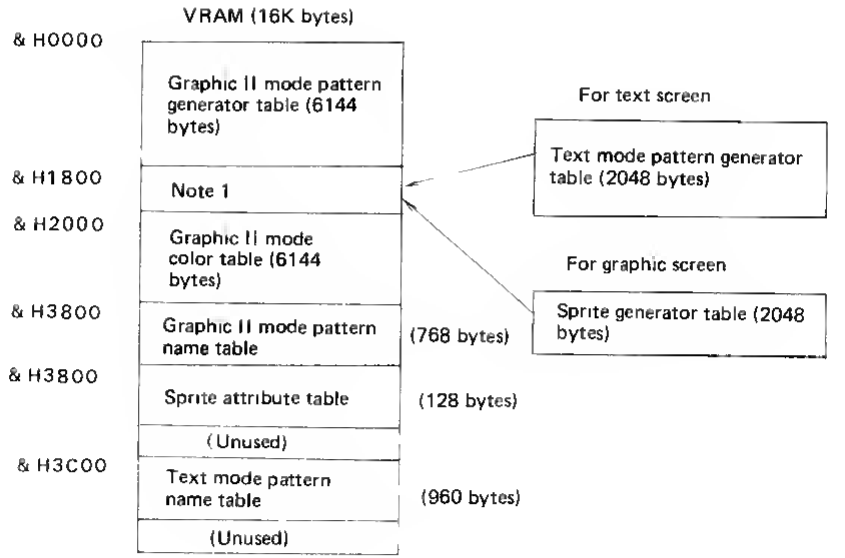
A character or symbol is generated by vertically aligned eight bytes (eight addresses). (See the explanation on the PATTERN statement.)

Addresses beginning with &H2000 in the color table have one-to-one correspondence with those beginning with &H0000.

Write data in eight bytes having addresses &H0650 to &H0657

The color table begins at &H2000. See the explanation on the COLOR statement for the color numbers

VRAM MAP



Pattern generator table (text screen)

```
1000 C=255
1010 FOR A=&H1800+32*8 TO &H1800+32*8+
7:VPOKE A,C:NEXT
1020 IF INKEY$=" " THEN C=0
1030 IF INKEY$="Z" THEN C=255
1040 GOTO 1010
```

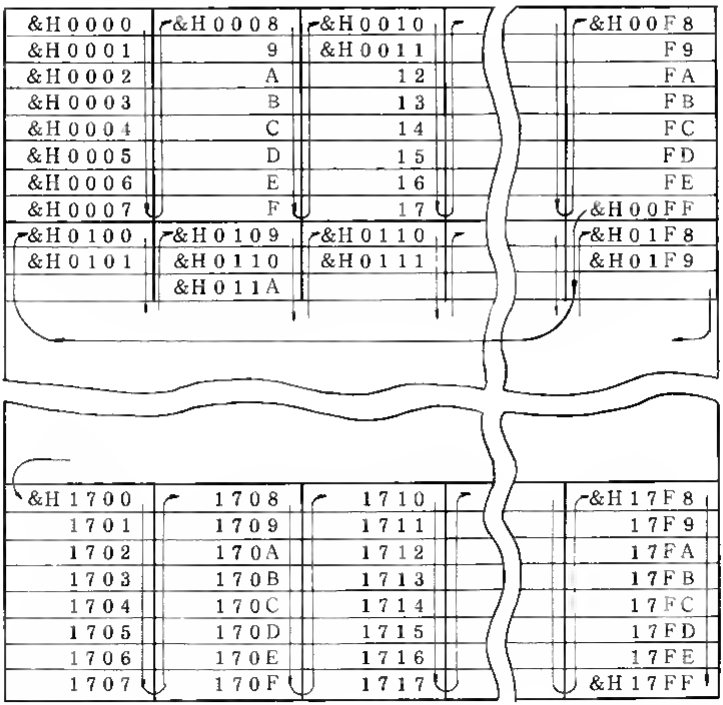
C=255 indicates a character code.

Sprite generator table (text screen)

```
2000 SCREEN 2,2:CLS
2010 FOR A=&H1800+32*8 TO &H1600+32*8+
7:VPOKE A,255:NEXT
2020 FOR M=0 TO 3
2030 MAGM
2040 FOR W=0 TO 100 : NEXT W
2050 SPRITE 0,(100,100),32,8
2060 NEXT M
2070 GOTO 2000
```

Screen top-left corner (starting address)

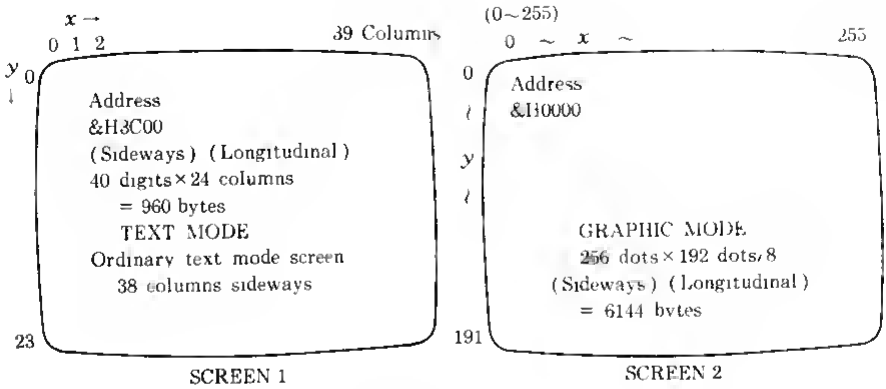
Screen top-right corner



Screen bottom-left corner

Screen bottom-right corner (ending address)

VPOKE ADDRESS ASCII DATA (Text mode)



Part of VRAM MAP

The address calculations on the text screen are carried out as follows.

$$\text{Address (text)} = y * 40 + x + \&H3C00$$
$$\text{where } (x = 0 \sim 39,\ y = 0 \sim 23)$$

For the data to be sent, the ASCII code of the corresponding character is applicable (32~255 in decimal numerals and &H20~&HFF in hexadecimal numerals).

Note: As shown in the left figure above, the horizontal axis is deviated by 2 columns as compared to the ordinary text screen. Thus, the display position defined by CURSOR statement deviates from that defined by VPOKE, by about 2 locations in the horizontal direction.

VPOKE ADDRESS, DATA (Graphic mode)

Graphic address calculations are carried out as follows.

$$\text{Graphic address} = \text{INT}(y/8) * 256 + \text{INT}(x/8) * 8 + y \text{ MOD } 8$$
$$\text{where } (y \text{ is } 0 \sim 191 \quad x \text{ is } 0 \sim 255)$$

The address derived from the above calculations is the beginning address of 8 bits (dots) in the assigned horizontal direction. The assigned address is the x-INT $(x/8)$ bit location counting from the left of the beginning address.

The data to be sent are hexadecimal or decimal numerals displayed by the bit pattern in a horizontal row

Example

 ⇨ &H93 147 ,

Similarly, the color table address for graphic color assignment is derived from the addition of &H2000 to the above address. The data to be sent are natural numbers (0 ~ 255) of 1B (1 Byte). The upper 4 bits of these numbers converted into binary data are the assigned color number and the lower 4 bits, the background color number. (The addresses of the graphic pattern generator table and color table respectively corresponds at 1 : 1)

Graphic color table address

$$= \mathrm{INT}(y/8) * 256 + \mathrm{INT}(x/8) * 8 + y \ \mathrm{MOD}\ 8 + \&H2000$$

Where y is 0 ~ 191
x is 0 ~ 255

Color data = Assigned color No * 16 + background color No
(0 ~ 15) (0 ~ 15)

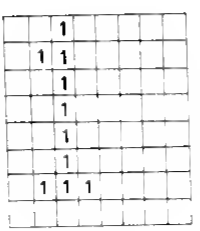
VPEEK

Use VPEEK with reference to VPOKE address. Program to read the content of the pattern generator table in VRAM

Example:

```
10 AD=&H1800+&H31*8     : REM The beginning address
20 FOR A=AD TO AD+7       of REM "1" pattern
30 DA=VPEEK(A)
40 PRINT HEX$(DA
50 NEXT A
```

```
20
60
20
20
20
20
70
00
```



CHAPTER 4 USE OF FLOPPY DISK

Floppy disk advantages

Because of a large memory capacity and a high accessing speed, floppy disks are mainly used as an external memory of computers. SC-3000 provides Supercontrol Station SF-7000 as one of its peripheral devices. The floppy disk drive incorporated in SF-7000 has significantly improved the SC-3000 capabilities.

The SC-3000 disk BASIC will facilitate disk operations. The disk BASIC is convenient for program saving and data filing, and especially for random file operations.

Disk BASIC

The disk BASIC provides disk operation commands and statements, in addition to the statements of SC-3000 Level III. Some commands and statements are used in different ways from those of SC 3000 Level III. Disk BASIC commands and statements which are frequently used are as follows.

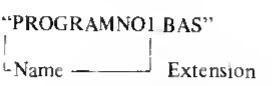
SAVE and LOAD	Program writing to an reading from disk
OPEN and CLOSE	File opening and closing
PRINT# and INPUT#	Writing to or reading from sequential file
EOF (function)	Detecting the end of sequential file
PUT# and GET#	Writing to and reading from random file
DISKI$	Reading directly from disk
MERGE	Merging programs
FILES	Displaying list of file names written in program

There are more disk operation commands. The major commands are explained below.

SAVE

The SAVE command saves generated programs on a disk. The disk is automatically searched for the saving location.

A file name must be assigned to the program to be saved. A file name must consists of eight or fewer characters, but when a period (.) is used, up to three characters can be added as an extension.



Since the name and extension are functionally the same, they can be assumed as one name consisting of up to 11 characters delimited by a period (.). The extension is used to identify the file contents. For example, a BASIC program is assigned an extension ".BAS" to identify it from programs in other languages.

If file name "BASIC" is assigned to the program saved on a disk, a FILES statement lists this name as "BASIC . ". Even if the assigned file name consists of 11 characters without a period a period is inserted in the name listed with a FILES statement If a new program is saved and its file name is the same as that of an existing file on the disk, the existing program is substituted with the new program When the new program name differs from an existing program name by only one character, the new program is saved on the disk, even if the contents are the same.

A file name may contain any kind of characters that can be displayed, except a period ( ) and double quotation mark ("). Small and capital letters are assumed to be different.

FILES

The names of program and data files are recorded in track 20 of the disk Use a FILES statement to check the files stored on the disk

Enter FILES CR

A FILES statement displays a list of all file names on the disk If the disk contains many files, the screen is scrolled up and te file names first displayed will disappear from the screen To suspend scrolling of the screen, press the space key.

When the object file name is found, press the BREAK key to stop file name display In this case, subsequent file names are not displayed

To load the program move the cursor to the left side of the file name displayed, enter LOAD, then press the CR key.

Files are classified into program and data files, which can be recognized by the existence of a period ( ) in the file name

```
"PROGRAM1 .BAS "    Program file
"ADRRESS  .BAS "
"DATA  NO1     "    Data file
"BOOKFILE DAT "
```

Data files are sequential or random files containing data

LOAD

This command reads the program saved on a disk to the SF-7000 memory. In this case, the file name used for saving must be used. If the file name is different, the file cannot be read. When a file name contains spaces, they are treated as characters. "TEL FILE.BAS" and "TELFILE .BAS" are different file names. If the file name is not known, list up file names with a FILES statement. When file names are listed with a FILES statement, the necessary file can be loaded only by moving the cursor to the associated file name, entering LOAD, then pressing the CR key. This is the simplest method for loading.

During loading, the disk keeps rotating at a high speed and the red lamp is on. When loading is completed, the red lamp goes off and the cursor appears again.

While the red lamp is on, never take out the disk, otherwise, loading fails and the records on the dik may be destroyed. Only program files can be loaded. Data files cannot be loaded, if it is attempted, an error occurs and error message "Bad file mode error" is displayed.

KILL

When many files are saved on a disk, it may be filled and have no free space to save more files. Unnecessary files can be erased from the disk with a KILL statement.

KILL "file-name extension" CR key
KILL "file-name " CR key

The KILL statement can erase only program files whose names contain a period ( ), it cannot erase data files whose names contain no period. For example, to erase a data file "DATAFILE TEL", insert a period between the file name and extension, then execute a KILL statement.

KILL "DATAFILE.TEL"

To insert a period, display the list of file names with a FILES statement, move the cursor to the space between the file name and extension, then enter a period ( ).

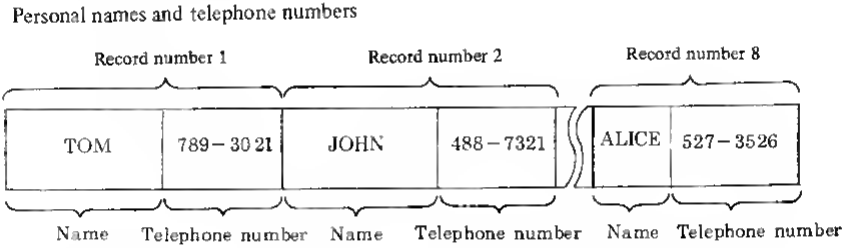
File management

Disk files are classified into sequential and random files. They have different characteristics and must be used correctly according to the purpose.

Sequential file

Data is sequentially stored in a sequential file, like recording music on magnetic tape. When data is sequentially entered, it is written on the disk, beginning at the first recording position. Data of an arbitrary length may be entered unless it exceeds the maximum character string length. The next data is written succeeding the previous data. Since the data length is not fixed, it is impossible to rewrite specific data later. This is similar to replacing one of songs recorded on magnetic tape. If a new song is recorded on an intermediate area of magnetic tape containing many songs, the preceding song may be erased at its ending part and the succeeding song may be erased at its beginning part. Likewise, if data is inserted in an intermediate area of a disk, the data recorded on this area is destroyed.

To avoid such a trouble in a sequential file, new data must be written after the last data.



As shown in the above figure, data for one person consists of two items: the name and telephone number. This is called a record. A record may contain more than one items.

A set of data is called a file and a file is assigned a name for management.

A sequential file can be easily used, though it is subject to some writing rules, so let's generate a sequential file first.

Generating a telephone directory

Generate a telephone directory in a sequential file. If possible, prepare a new disk. An old disk may be used, but a trouble may occur if the new data is mixed with old data, so it is better to use a disk only for the telephone directory.

A sequential file consists of the file name, file number, and items.

File name

When generating files to record telephone numbers, music, and birthdays of your friends, a file name must be assigned to each file.

File number

Data written on or read from a disk passes through the data buffer. The data buffer is a special area in the memory, it is also called a buffer memory or simply a buffer.
A buffer is a 256-byte area used to read or write one sector (256 bytes) on a disk.
There are buffers 0 ~ 8. buffer 0 is used for communications via the RS-232C interface.
Before writing or reading disk data, specify the file number to open the associated buffer. After writing or reading ends, close the file with a CLOSE statement.

Items

A telephone directory must have two items: the names and telephone numbers. If addresses are necessary, one more item must be used. A record consists of such items.
For a sequential file, record numbers need not be specified. The items written while the file is open are assumed to be records. When reading, the items for a record are output together.

You can learn how to generate a sequential file, using a telephone directory as an example. First, specify the necessary items.

File name : FS="TELEPHONE DIRECTORY"
Item 1 : NA$="NAME"
Item 2 : TL$="TELEPHONE NO"

Assign the name of FS="TELEPHONE DIRECTORY" to the file. This program assumed that the file name is FS.

```
10 REM ***** TELEPHONE NOTE *****
20 CLS
30 F$="TEL NOTE"
```

Generate a writing file

Use line 120 to open the disk file, and lines 130 and 150 to assign the name and telephone number to the character string variable. When END is specified as the name, the menu screen is displayed. In line 160, specify a PRINT# statement to open file 1 and write the contents of the character string variable to the disk. After writing, close the file with a CLOSE statement.

```
110 PRINT F$
120 OPEN F$ FOR OUTPUT AS #1
130 INPUT NA$
150 INPUT "TELEPHONE";TL$
160 PRINT #1,NA$;",";TL$
```

A telephone directory for many persons can be generated by repeating the step above, but this program enables only writing.

General a reading program

Also when reading data from the disk, the file must be opened by specifying the file name and file number. If a large volume of data has been written, it cannot be displayed at a time, therefore, only the necessary data is displayed in this case

To read data, enter only the object name to collate it with the data on the disk. It is better to display the object data, if found.

```
200 REM ----- READ -----
210 PRINT:F$:PRINT:PRINT
220 OPEN F$ FOR INPUT AS #1
230 INPUT "SEARCH NAME      ";SN$
250 IF EOF(#1) THEN 500
260 INPUT #1,NA$,TL$
270 IF SN$ <>NA$ THEN 260
280 PRINT:PRINT NA$:PRINTTL$:PRINT
290 IF INKEY$="" " THEN 500
```

Use line 220 to open the file, line 230 to assign the object name to the character variable, line 260 to read only a set of data from the disk, and line 270 to collate the character variable entered with that read from the disk

If the character strings do not match, the program returns to line 260 to read another data, and collates it again. Thus, the program repeats reading data from the disk until the object name is found, when found, line 280 displays it After displaying it, close the file with a CLOSE statement.

Append data

Note that appending new data to an existing telephone directory is not writing If additional data is written by line 100, the program assumes that a new file has been generated

Since an APPEND statement is used to append data, another subroutine must be prepared. Also in this case, open the file to declare appending.

```
300 REM ----- APEND -----
320 OPEN F$ FOR APPEND AS #1
330 INPUT "NAME      ";NA$
340 IF NA$="END" THEN GOTO 500
350 INPUT "TELEPHONE";TL$
360 PRINT #1,NA$,TL$
370 GOTO 330
```

After declaring appendance by opening the file in line 320, the procedure to be performed is the same as that for writing. The following program is a linkage of the sample programs above. This program gives the basic form for a sequential file

A sequential file is sequentially read from the beginning and cannot be read backward; that is, previous data cannot be read immediately after reading data near the end. This is a disadvantage of a sequential file. In the case above, once close the file, reopen it, then read it from the beginning.

Before writing your programs, you must understand these characteristics of a sequential file.

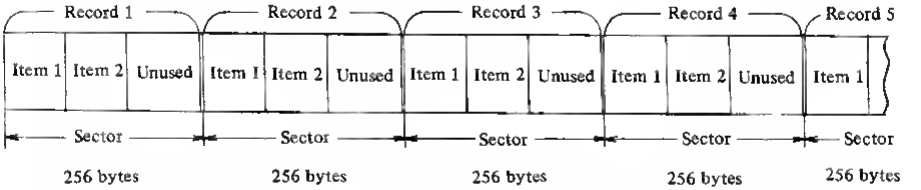
```
10 REM ***** TELEPHONE NOTE *****
20 CLS
30 F$="TEL NOTE"
40 GOTO 520
100 REM ------ WRITE ------
110 PRINT F$
120 OPEN F$ FOR OUTPUT AS #1
130 INPUT "NAME      ";NA$
140 IF NA$="END" THEN 500
150 INPUT "TELEPHONE";TL$
160 PRINT #1,NA$,TL$
170 GOTO 130
200 REM ------ READ ------
210 PRINTF$:PRINT:PRINT
220 OPEN F$ FOR INPUT AS #1
230 INPUT "SEARCH NAME      ";SN$
250 IF EOF(#1) THEN 500
260 INPUT #1,NA$,TL$
270 IF SN$<>NA$ THEN 260
280 PRINT:PRINT NA$:PRINTTL$:PRINT
290 IF INKEY$="" THEN 500
295 GOTO 290
300 REM ------ APEND ------
310 OPEN F$ FOR APPEND AS #1
320 INPUT "NAME      ";NA$
340 IF NA$="END" THEN GOTO 500
350 INPUT "TELEPHONE";TL$
360 PRINT #1,NA$,TL$
370 GOTO 330
400 CLOSE
500 CLS
510 PRINT "WRITE ..... 1"
520 PRINT "READ ..... 2"
530 PRINT "APPEND ..... 3"
540 INPUT "PUSH 1-3 ";A
550 ON A GOTO 120,220,320
```

When END is entered instead of the name, the program displays the menu screen.

Random file (direct file)

A random file is advantageous in arbitrary writing, reading, and rewritting A sequential file enables addition, not rewritting, while a random file has data areas of a fixed size in which data can be arbitrarily rewritten

A random file must have a file name, record numbers for specifying data areas, and data items



A random file mainly differs from a sequential file in that its data areas have a fixed size A sequential file has variable-length data areas because the area size vary according to the data length, while a random file has fixed-length data areas The data areas are called records One record occupies a sector which can store character data of up to 255 bytes

When data to be stored has more than one item, the number of items and the number of characters in each item are used to decide how to store the data When there are five items, each item may have 50 characters. For personal names and telephone numbers, an item may have 10 to 15 characters and unused areas are allocated to items having many characters The program determines the number of characters to be contained in an item When the entered characters exceeds the determined number, the excessive characters are discarded.

When generating a random file, you must carefully decide how to sotre data If this is incorrectly set, all data may not be stored

Unlike for a sequential file, the number of records in a random file must not exceed the maximum number of sectors on the disk For a sequential file, the number of records varies according to the data sizes, but this file can be efficiently used because it sores data without unused areas

Generating a random file

To open a random file, specify its file name and file number Check the file size with an LOF function Assign data to a character variable and write it onto a disk using a PUT statement In case of a random file, specify the item, data storage area, and the number of characters in a put statement then close the file. Random file data can be read in the same way as that for writting, except that data read with a GET statement is assigned to a variable, then displayed with a PRINT statement

Program explanations

```
70 REM ----- WRITE ------------
80 OPEN "TEL NO. " AS #1
90 M=LOF(1)
100 PRINT"DATA COUNT";M
110 INPUT "RECORD NO. ";R
120 IF R=0 THEN GOTO 190
130 REM **** GET ***
140 INPUT "NAME  ";A$
150 INPUT "TEL   ";B$
160 REM ****** PUT *****
170 PUT#1,R;A$,0,20;B$,20,20
180 GOTO 110
190 CLOSE:GOTO 10
```

Line 110 opens the file, line 120 checks the number of records which is displayed with line 125, and line 130 specifies the record number. If a record containing data is specified in line 130, the record data is updated, therefore, line 130 must be specified carefully, using the number of records obtained by the LOF function.

Line 140 is used to quit the write mode; when 0 is specified, writing is terminated.

Lines 190 ~ 220 are used to specify data which is written on the disk with line 240.

In the PUT statement, #1 specifies the file number and R specifies the record number. The numeral succeeding the character variable is an offset indicating the writing position in the sector and the next numeral indicates the number of characters (length). These values define the maximum number of characters entered; therefore, they must be carefully decided during file design. BS succeeding the semicolon (;) indicates the second item in the same sector; it defines the number of items. After line 250, return to line 130 to write the next data using new record numbers.

The data can be read from the disk in mostly the same way. The GET statement in line 360 must have the same format as that used for writing. If it has a different format, the read data may not be the same as the written data.

Program for reading

```
200 REM ----- READ ------------
210 OPEN "TEL NO. " AS #1
220 M=LOF(1)
230 PRINT"DATA COUNT";M
240 INPUT "RECORD NO. ";R
250 IF R=0 THEN GOTO 310
260 REM **** GET ***
270 GET#1,R;A$,0,20;B$,20,20
280 PRINT "NAME  ";A$
290 PRINT "TEL   ";B$
300 GOTO 240
310 CLOSE:GOTO 10
```

The following program is generated by linking the two programs above. You can write your own object programs based on this program.

```
10 REM ------- TELEPHONE NOTE ----
20 PRINT"TELEPHONE NOTO "
30 PRINT"WRITE ..... 1"
40 PRINT"READ  ..... 2"
60 INPUT " 1~3 PUSH ANYKEYS";KY
70 ON KY GOTO 100,300
100 REM ----- WRITE -----------
110 OPEN "TEL NO. " AS #1
120 M=LOF(1)
125 PRINT"DATA COUNT";M
130 INPUT "RECORD NO. ";R
140 IF R=0 THEN GOTO 260
180 REM **** GET ***
190 INPUT "NAME  ";A$
200 INPUT "TEL   ";B$
210 REM ****** PUT ****
240 PUT#1,R;A$,0,20;B$,20,20
250 GOTO 130
260 CLOSE:GOTO 10
300 REM ----- READ ------------
310 OPEN "TEL NO. " AS #1
320 M=LOF(1)
325 PRINT"DATA COUNT";M
330 INPUT "RECORD NO. ";R
340 IF R=0 THEN GOTO 420
350 REM **** GET ***
360 GET#1,R;A$,0,20;B$,20,20
380 PRINT "NAME  ";A$
390 PRINT "TEL   ";B$
410 GOTO 330
420 CLOSE:GOTO 10
```

The above two programs give the basic forms, programs for practical use can be coded by modifying them.

CHAPTER 5 COMMANDS, STATEMENTS, AND FUNCTIONS

This chapter explains the disk BASIC commands, statements, and functions in the alphabetical order. To know the classification, see the list at the end of this manual.
In the explanation, an expression indicates a function or variable expression, which may be different from arithmetic expressions. The explanation on each command, statement, or function consists of Function, Format, Description, Notes, and See Also.

Function — Explains the function of the command, statement, or function.

Format — Gives a sample format. The parameters and variables are detailed in the description. Also see sample programs for actual coding.

Example: PRINT A → PRINT variable-name

In the format, complicated symbols are not used for easy understanding. You can input the format as shown.

Description — Describes the function in details. Also read the explanation on the associated commands, statements, and functions, if any.
Note that, for programming, a comma (,) differs from a period (.) and a colon (:) differs from a semicolon (;).
A minus sign (-) also differs from a hyphen and longvowel sign in Kanas.

Example — Understand how commands and statements run, using this example and sample programs.

Note(s) — Gives programming notes.

See Also — Some commands are used in a combination. Such commands and associated commands are given here.

Error messages are listed in the Appendix. When an error message is displayed, see this list to locate the error.
The sample programs given in this manual can be applied to your own programming by modification and linking.

Command	AUTO

Function Generates line numbers automatically

Format AUTO start line number, increment

AUTO	Automatically generate line numbers starting from 10 with an increment of 10
AUTO 100, 50	Automatically generate line numbers starting from 100 with an increment of 50

Description This command is useful when inputting a program with a predetermined structure.

It automatically types onto the screen the next line number whenever you hit the [CR] key.

Press the [BREAK] key to stop this command.

You can choose any numbers for the start line number and the increment although the line numbers must be in the range less than 65536.

Example :

```
AUTO
10
20

AUTO 100
100
110

AUTO 10,20
10
30
50
```

Command	BOOT

Function: Re-loads the system programs (the BASIC interpreter and so on) from disk

Format: BOOT

Description: This command interrupts the BASIC interpreter, deletes it from RAM and initiates the IPL (Initial Program Loader).
If your system disk is in the drive at the time of the execution of this command, reload will begin immediately.

Note: If some programs constructed with the DISK BASIC remained in memory, they are deleted by this command.

Example:

```
BOOT
```

See also: UTILITY command B

Command	CLOAD	(cassette load)

Function Loads into memory programs saved on cassette

Format CLOAD "filename"

Description You must use CLOAD to load programs from cassette and not LOAD which is meant for disk.
If there is just one program saved on your cassette, then you need not specify "filename," but if there is more than one, then you must specify "filename "
If there is a mismatch between the name you supplied to this command and the one on the cassette, then the message
Skip "filename"
will be displayed and the program you intend to load will not be loaded
Note here that even the case (upper/lower) difference counts as a mismatch
In case the program you intend wouldn't be loaded, push the reset key and repeat the entire process of the load once more
If you have registered the count on the tape recorder counter, then spin the tape up to the place marked by the count and start loading from that place

Example

```
CLOAD"
* Loading start
 Found
* Loading end
```

Command COMLOAD (communications load)

Function: Receives programs via the RS-232C interface

Format: COMLOAD

Description: This command lets you receive into memory programs sent thru an RS-232C interface line.
Communications thru the RS-232C can be performed either by connecting two computers directly or by utilizing an existing telephone communications line with the aid of an acoustic coupler
The RECEIVE lamp will be kept lit during the transfer.

Example:

```
COMLOAD
```

Command COMSAVE (communication load)

Function: Sends programs via the RS-232C interface

Format: COMSAVE

Description: Use this command to send programs in memory to other computers via an RS-232C interface line.
To this end either connect the two computers directly or utilize an acoustic-coupler.
In the latter case, setting the telephone receiver into the acoustic-coupler will send the program under the form of acoustic signals
The SEND lamp will be kept lit during the transfer

Example:

```
COMSAVE
```

Command	CONT	(continue)

Function: Resumes execution of programs previously interrupted by the BREAK key or by the STOP statement

Format: **CONT**

Description: A running program can be interrupted either by a STOP statement placed in the program or by hitting the BREAK key to see, for example, the content of a variable. To see the content of a variable, type the direct command PRINT "variable name" followed by CR.
Type CONT followed by CR to resume execution.
Note that an interrupted program cannot be resumed its execution if you modify or add some new lines to it during the interrupt. In such cases, the message

Can't continue error

will appear on the screen

Example:

```
LIST

10 FOR I=1 TO 9
20 FOR J=1 TO 9
30 PRINT I*J;
40 NEXT J:PRINT
50 STOP
60 NEXT I

RUN
 1  2  3  4  5  6  7  8  9
Break in 50
PRINT I,J
 1                    10
Ready
CONT
 2  4  6  8  10  12  14  16  18
Break in 50
```

Command	CSAVE	(cassette save)

Function Saves the program in memory onto cassette

Format CSAVE "filename"

Description This command is equivalent to SAVE except programs are saved onto cassette
Filename is limited up to 16-characters' long and must be enclosed in double quotes (") as in
"BASIC"
Record the filename lest you forget it The CLOAD command will not be able to find the program you want unless you supply the correct filename to it
Also it is a good idea to check and record the start and the end values of the tape recorder counter when you execute this command
Do not forget to check whether the program has been correctly saved with the VERIFY command

Example :

```
CSAVE"
* Saving start
* Saving end
```

See also CLOAD, VERIFY

Command	DELETE

Function: Deletes parts of a program from memory

Format: DELETE start line number – end line number
deletes the lines between the indicated line numbers inclusively

DELETE – line number	Deletes lines from the youngest up to the one indicated by the line number
DELETE line number –	Delete lines from the one indicated by the line number up to the oldest
DELETE line number	Delete only the line indicated by the line number

Description: Use this command to delete a group of lines at the same time, though there are some other methods to delete parts of a program described below.

* Type the line number of the line you want to delete and hit the CR key
* Place the cursor just after the line number of the line you want to delete, keep pressing the space key until the line except the number is erased from the screen, then hit the CR key

Although you can erase lines from the screen with the space key or with the INS/DEL key, they remain in memory until you hit the CR key. The LIST command will display the lines you have erased from the screen without hitting the CR key.

Example:

```
DELETE 180-220

DELETE -250

DELETE 500-

DELETE 100
```

Command	FILES

Function Displays on the screen the list of file names of programs on disk

Format FILES

Description Use this command to see what kind of programs are there on disk, since there can be many programs saved on disk at the same time. It will display on the screen all the file names of programs on disk.

If you have more than one screen-full of files, the earlier files will slip off the screen while you watch.

Use the space key to interrupt the flow, and stroke it once more to continue the flow.

Press the **BREAK** key to stop the command entirely, since **LOAD**ing can be done only after this command has been completed or stopped entirely.

Example

```
FILES

"SAMPLE 1.bas"
"SAMPLE 2.bas"
"SAMPLE 3.bas"
"DEMO   1.bas"
"DEMO   2.bas"
"DEMO   3.bas"
"DEMO   4.bas"
"SOUND  1.TST"
"SOUND  2.TST"

 [illegible] files free
```

Command	LFILES

Function Outputs to the printer the name list of files on disk

Format LFILES

Description Since there can be many files saved on disk at the same time, this command is used to printout all the names (not contents) of files now saved on disk, letting you know what kind of programs are there on disk.
The printout can be used as an index.

Example

```
LFILES
```

Command	LIST

Function: Displays on the screen contents of the program in memory either partly or entirely

Format: LLIST

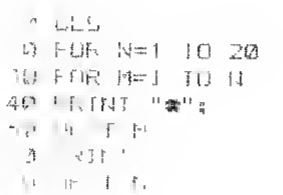
where the '–' (minus sign) can be replaced by a ' , ' (comma)

LLIST	Display the entire program
LLIST line number	Display only the line indicated by the line number
LLIST line number – line number	Display the lines between the indicated line numbers inclusively
LLIST line number –	Display from the line indicated by the line number to the end of the program
LLIST – line number	Display from the start of the program up to the line indicated by the line number

Description: Use this command to look at or modify the program in memory.
Big programs will scroll off the screen while you watch.
Hit the space key to interrupt the flow, and hit it once more to continue the flow.
Hit the BREAK key if you want to abandon the display.
You can modify the content of your program thus displayed (screen edit).

Example:

```
LIST
```



Command	LLIST

Function: Outputs to the printer contents of the program in memory, either partly or entirely

Format: LLIST

LLIST	Print the entire program
LLIST line number	Print only the line indicated by the line number
LLIST line number – line number	Print the lines between the indicated line numbers inclusively
LLIST line number –	Print from the line indicated by the line number to the end of the program
LLIST – line number	Print from the start of the program up to the line indicated by the line number

Description: Use this statement to printout parts or the entire contents of your programs, for checking or for preservance.

Example:

```
LLIST

LLIST 100

LLIST -100

LLIST 100-

LLIST 100-200
```

Command	LOAD

Function: Loads programs from disk

Format: LOAD "filename"

Description: Display the names of programs on disk with the FILES command. Then move the cursor to the name of the program you want to load. Type LOAD and hit the [CR] key.
The program you indicated will be loaded into memory.

Example :

```
FILES

      "SAMPLE  1.bas"
      "SAMPLE  2.bas"
      "SAMPLE  3.bas"
LOAD  "DEMO    1.bas"
      "DEMO    3.bas"
      "DEMO    2.bas"
      "DEMO    4.bas"
      "SOUND   1.TST"
      "SOUND   2.TST"

49K bytes free
```

Command	MAXFILE

Function: Declares the maximum number of files that can be opened simultaneously for processing.

Format: MAXFILE n n is an integer in the range 0 thru 8 exclusive

Description: The BASIC interpreter initially sets the maximum number of files that can be opened simultaneously to 3
Use this command if you need open more than three files at the same time.

Note: This command initializes all the memory areas including the BASIC program area. Consequently if a program happened to be in memory, you must first save it onto disk or cassette. The size of the free area will change after the execution of this command.

Example:

```
MAXFILE 4
```

Command	MARGE

Function Merges (joins) a program on disk with the program in memory

Format MARGE "filename"

Description This command merges a program on disk with that in memory and thereby creates one single program in memory

Note that line numbers used in one program must not appear in the other Use RENUM for this purpose

For example, if the program in memory has line numbers 10 thru 500, number the program on disk to be merged starting from 510

If a same line number appeared in both programs, the contents of the line under that line number of the merging program would override the other

Example

```
FILES

        "SAMPLE  1.bas"
        "SAMPLE  2.bas"
        "SAMPLE  3.bas"
        "DEMO    1.bas"
        "DEMO    2.bas"
        "DEMO    3.bas"
        "DEMO    4.bas"
        "SOUND   1.TST"
MERGE   "SOUND   2.TST"

47K Bytes free
```

Command NEW

Function: Deletes programs and resets variables in memory

Format: NEW

Description: If you input lines of a program while there is some other program still in memory, they get mingled up and may lead to some unexpected result or error. You must execute this command to delete some previous program from memory whenever you input a new program.
To see whether there is some program still in memory, use the LIST command. If there is one, delete it with this command.

Example:

```
NEW
```

Command NEWON

Function: Sets the start address for the BASIC program area

Format: NEWON start address

Description: This command allocates areas of memory starting from the given address to BASIC programs, arrays, variables and so on.

Note: You cannot set the address within the area for the BASIC interpreter, work area, nor in the area higher than the address previously set by the LIMIT statement.
This command, like the NEW statement, deletes state programs currently in memory.

Example:

```
NEWON &HC000
```

Command	RENUM	(venumber)

Function Re-sequences the line numbers of a program

Format **RENUM** new line number, current line number, increment

Description RENUM followed by the [CR] key will re-sequence the line numbers starting from 10 with an increment of 10. The line numbers appearing in a GOTO statement or in a GOSUB statement will be adjusted accordingly.
If you omit increment, it defaults to 10

Note If there is a line number that does not exist in the program and yet is referenced in one of GOTO, GOSUB, IF-THEN and RESTORE statements, then this command will cause the Undef'd line number error.

Example

```
RENUM

RENUM 100

RENUM 300,200

RENUM 300,200,50
```

Command	RUN

Function: Starts execution of a program

Format:

RUN	Starts execution from the beginning of the program in memory
RUN line number	Starts execution from the line specified by line number
RUN filename	Starts execution of the named program after loading it from disk

Description: Though SC-3000 has a function key for it, this statement is still useful if you want to execute a program from a given line or execute one of two programs in memory demarcated by separate line numbers.

Example :

```
LIST

100 A=100
110 PRINT A

RUN
 100
Ready

RUN 100
 0
Ready
```

Command	SAVE

Function Saves the program in memory onto floppy disks

Format SAVE "filename.extension"

Description Filename is a name you give to a program to somehow remember its function. And a program can be saved only after you have christened it.
Either a complete program or a program under development can be saved.
When you create a big program, you can temporarily leave the job by saving your intermediate result, and later resume the job by loading it back.
Filename is limited to up to 8 characters optionally followed by a '.' (period) and a 3 character extension.
If you save a program under a name, and if there is a program under that name on disk, then the program on disk will be replaced by the newly saved one. If you make some modification to a program that was previously saved on disk, then choose the same name as that of the program. But if you want to save the modified program separately, then choose partly different name from the original one.
This is because, on disk save, the place to where programs go is selected from the given floppy disk unit.

Example :

```
SAVE "SAMPLE.TST"
```

See also : LORD , FILES

Command	UTILITY

Function Enters the disk utility program which in turn accepts the following commands described below

Format UTILITY CR

Description UTILITY commands

F Format disks
C Copy disks
B Do a BOOT

F Disk formatting

A new disk can be used only after you have formatted it. Type this command. Set your disk into the drive and type F followed by CR.

Don't do anything before the cursor appears on the screen, since interruption of this command sometimes means disk destruction.

Note also that if you format a disk with some programs still in it, the programs are deleted.

C Sometimes, though rarely, a disk may be damaged and become useless with all its plastic coverage.

You are recommended to take copies of your important programs by this command against such disaster.

B Do a BOOT

How to copy your disks

Press the C key followed by the CR.

Set the floppy disk you want to have a copy of (SOURCE DISK), then press the space key.

Ten tracks of data will be read into the drive.

Pressing the space key at this time will start the copy. The copy will take some time. Don't interrupt the disk unit while copying since it will destroy the copy.

If you use a disk with some programs already in it, those programs will be replaced by the newly copied ones.

Repeat the above procedure 4 times to complete the copy on one side.

The message

copy complete

will appear on the screen on completion of the copy.

Copy uses a different format than that used in save. So although the BASIC system cannot be saved onto disks, it can be copied there.

Take a backup of your BASIC system by copying it to some disk.

Set also BOOT

Command	VERIFY

Function Compares the program saved on cassette and the program in memory

Format VERIFY "filename"

Description This command checks whether the program in memory has been correctly saved onto cassette.
Rewind the tape upto where you started the save.
Type
VERIFY "filename"
followed by pressing down the [CR] key.
Then push the play (LOAD) key on the tape recorder.
If no difference is found between the program in memory and the program saved on cassette, the message

Verify end

will appear on the screen.
If the message would not appear, push the reset key to break the command and restart from the save

Example

```
VERIFY
* Verifying start
 Found
* Verifying end
```

See also SAVE, LOAD

Statement	BCIRCLE

Function: Erases lines or circles drawn on the screen

Format: BCIRCLE (X, Y), radius, ratio, starting point, end point, BF

Description: The statement is used in the same way as the CIRCLE statement to erase desired area, though you cannot specify color to this statement.
The color corresponding to bit "0" is chosen to erase the area.

Example:

```
10 SCREEN 2,2:CLS
20 FOR R=5 TO 1 STEP -1
30 CIRCLE(128,90),R*10,R,1,0,1,BF
40 BCIRCLE(128,90),R*9,,1,0,1,BF
50 NEXT R
60 GOTO 60
```

See Also: CIRCLE, COLOR

Statement	BEEP

Function Generates a beep sound

Format **BEEP** *n*

Description *n* must be in the range 0 thru 2

BEEP	Beep
BEEP 0	Stop beeping caused by BEEP 1
BEEP 1	Keep beeping
BEEP 2	Generate sound like peep poop

Example

```
10 DIM A$(12)
20 FOR N=0 TO 12
30 READ A$(N)
40 PRINT A$(N);
50 BEEP
60 NEXT N
70 DATA H,O,M,E," ",C,O,M,P,U,T,E,R
RUN
HOME COMPUTER

Ready
```

Statement	BLINE

Fungion: Erases by line or rectangle

Format: **BLINE** (X1, Y1) – (X2, Y2)
BLINE (X1, Y1) – (X2, Y2), BF

Description: Colors cannot be specified to the BLINE statement. The color chosen is the color of the background at the time of execution of this statement.
The color of the background corresponds to bit "0". The BF specification will erase the rectangular area determined by (X1, Y1) and (X2, Y2).

Example:

```
10 SCREEN 2,2:CLS
20 FOR R=5 TO 1 STEP -1
30 CIRCLE(128,90),R*10,R,1,0,1,BF
40 BCIRCLE(128,90),R*9,,1,0,1,BF
50 NEXT R
60 GOTO 60
```

See Also: LINE, COLOR

Statement	CALL

Function: Calls machine language subroutines

Format: CALL start address

Description: Since machine language programs are placed outside the BASIC program area, you must use this statement to call a machine language subroutine.

Example:

```
10 LIMIT %HDFFF :CLS
20 FOR A=%HE000 TO %HE082
30 READ D$:D=VAL("%H"+D$)
40 POKE A,D
50 NEXT A
60 CALL %HE000
70 CURSOR 0,2:PRINT TAB(30*RND(1));"A"
80 GOTO 60
100 DATA F3,C5,D5,E5,F5,06,16,0E
110 DATA 00,CD,3D,E0,CD,58,E0,EB
120 DATA 21,83,E0,0E,27,CD,66,E0
130 DATA 77,23,0D,20,F8,EB,7D,C6
140 DATA 28,6F,30,01,24,CD,6D,E0
150 DATA 21,83,E0,0E,27,7E,CD,7D
160 DATA E0,23,0D,20,F8,10,D0,F1
170 DATA E1,D1,C1,FB,C9,C5,D5,26
180 DATA 00,68,29,29,29,54,5D,29
190 DATA 29,19,16,00,59,19,11,00
200 DATA 3C,19,D1,C1,C9,DB,BF,C9
210 DATA F5,CD,55,E0,7D,D3,BF,7C
220 DATA E6,3F,D3,BF,F1,C9,00,00
230 DATA 00,00,DB,BE,C9,F5,CD,55
240 DATA E0,7D,D3,BF,7C,E6,3F,F6
250 DATA 40,D3,BF,F1,C9,00,00,00
260 DATA D3,BE,C9
```

See also: **LIMIT**

Statement	CIRCLE

Function Draws circles around given points

Format CIRCLE (X, Y), radius, color, ratio, starting point, end point, BF

Description The statement draws a circle around the given point (X, Y). The arguments to this statement are explained as follows:

Radius: The scale for this quantity is measured in pixels (dots). If the length of the diameter go beyond the maximum value allowed for the coordinate, the part coming outside the coordinate will be drawn as a straight line.

Color: Specified by the color code

Radio: Ratio of diameter to the horizontal axis explained as follows.

Is equal to 1 The ratio is 1 to 1 and the circle drawn will be a true circle.

Is less than 1 And ellipse will be drawn with its horizontal diameter greater than the vertical diameter.
The allowable number of decimal places to the left of the decimal point is restricted to 1 (0.1, 0.2, . . . 1)

Is greater than 1 An ellipse with its vertical diameter greater than the horizontal diameter. The allowable values are 1.1, 1.2, . . up to 1.

Starting point: Just imagine a clock. The circumference of any circle is so measured that the number 0 corresponds to the position the small hand points at 3 o'clock, and hence the number increases clockwise along with the circumference up to 1 which finally comes to overlap with the starting point.
You can start drawing beginning from any point on the circle by specifying a decimal fraction between 0 and 1.
The fraction can be up to two decimal places.

End point: Can be any decimal fraction between 0 and 1 inclusive. The number 1 corresponds to the position the small hand points at 3 o'clock.
Supply the values like 0.25, 0.75 instead of 1.25, 1.75 since values greater than 1 will draw circles past the starting point.

B

Specifying B alone will draw line segments connecting the center of the circle to the starting and the end points.

BF

Specifying F in addition to B will paint the region drawn by the B specification.

Omitting arguments

The arguments to the CIRCLE statement can be omitted except the coordinate of the center and the radius.
If you omit color specification, the color employed by some previous statement will be used.

If you omit ratio, it defaults to 1.

If you omit the starting point specifiation, it defaults to 0.

If you omit the end point specification, it defaults to 1.

You cannot specify F without specifying B.

You need type only those arguments you need if you want to omit every arguments coming after a certain argument.
If you want to omit those arguments the positions of which come between other arguments, you need supply commas to indicate you have omitted those arguments (see example below).

Example

CIRCLE (X, Y), 50
CIRCLE (X, Y), 50, 8, , , , BF

```
10 SCREEN 2,2:CLS
20 FOR X=50 TO 200 STEP 5
30 CIRCLE (X,90),60,1,1,0,1
40 NEXT
50 FOR X=50 TO 200 STEP 5
60 BCIRCLE (X,90),60
70 NEXT
```

See Also

BCIRCLE, COLOR

Statement	CLOADM

Function: Loads machine language programs from cassette

Format: CLOADM "filename", load start address

Description: This statement loads the machine language program on cassette indicated by the "filename" onto memory. If you specify load start address, load will start from that address
If not, it will start from the address as previously indicated by CSAVEM
If you omit "filename", the first program encountered on cassette during the load will be loaded

Note: The filename must be the one you christened on save, otherwise the message "Skip" will be printed and no load will be done.

Example :

```
CLOADM
* Loading start
 Found OBJ: HEX DATA
* Loading end
Ready
```

Statement **CLOSE**

Function: Closes the specified file

Format: CLOSE #file descriptor, #file descriptor

Description: This statement closes the file indicated by the file descriptor. The closed files can be re-opened specifying to the OPEN statement the same or other file descriptors.
This statement can take more than one argument to close multiple number of files at the same time.
If you omit the file descriptor, all files currently open will be closed.
Any file, once opened, must be closed using this statement after you are satisfied with the processing of the file. All currently open files are closed not only by this CLOSE statement, but also by the END statement, the NEW statement, or by hitting the BREAK key.

Example:

```
10 OPEN "DATA      .     " FOR INPUT AS #1
20 INPUT #1,A$
30 CLOSE
```

See also: OPEN

Statement	CLS	(clear screen)

Function: Clears the currently active part of the screen

Format: CLS

Description: This statement erases everything, programs and results displayed by the previous run of some programs, from the current window (the part of the screen which is currently active). No other part of the screen other than currently active window will be affected by this statement

Example :

```
SCREEN 2,2:CLS
Ready
```

```
10 CLS
20 FOR E=0 TO 100
30 PRINT I;
40 NEXT E
```

Statement	COLOR

Function Sets color on the screen.

Format For the text window.

COLOR color code for character, color code for background for the graphics screen

For the graphics screen

COLOR c1, c0, (X1, Y1) – (X2, Y2), cb

Description c1: The color corresponding to bit "1" (clor for characters and lines)

The color applies to

* Characters printed by the PRINT statement
* Points or lines drawn by the PSET, LINE or CIRCLE statements
* Areas to be painted by the BF specification to the LINE or BLINE statement

c0. The color corresponding to bit "0" (background color)

Which applies to

* The window and the background after execution of the CLS statement
* The part with bit "1" reset by the PRESET, BLINE or BCIRCLE statements.

(X1, Y1) – (X2, Y2)

Paint inside the rectangle having the segment connecting (X1, Y1) and (X2, Y2) as its diagonal

The c0 argument must be specified.

cb: Color of the backdrop

(Equivalent to "transparency")

The backdrop is the upper and the lower margins of the screen in which neither characters or symbols, nor points or lines can be drawn.

"Transparency" corresponds to the color of these margins.

Each color has a code associated with it.

Color Code Table

Color code	Color	Color code	Color	Color code	Color
0	Transparency	6	Dark red	12	Dark green
1	Black	7	Light blue (cyane)	13	Mazenta
2	Green			14	Grey
3	Light green	8	Red	15	White
4	Dark blue	9	Light red		
5	Light blue	10	Dark yellow		
		11	Light yellow		

Note: The unit of area with which color can change from one to another consists of a horizontal row of 8 successive pixels (pixel is equivalent to picture element, which is the smallest dot of which characters or figures are comprised). Any area consisting of a successive row of 8 pixels can contain up to 2 colors including the color for the background, which means color for points or lines cannot vary within the area. And if you specify 3 different colors to paint the area, the entire area will be painted by the 3rd color specified.
Remind this fact when you use the LINE, CIRCLE or the PSET statement.
The above mentioned units are not placed arbitrarily on the screen. On any one line of the screen, the first unit consists of from 0 to 7th pixels, next from 8 to 15 pixels, and so on.

Additional Information: You can find in the explanation of the graphics mode for the SC-3000 those words such as pixel, dot and bit.

A pixel is the least unit of point used to draw figures in the graphics mode.
A bit is the least unit your computer can understand.
A dot is a least unit for drawing pictures under a certain condition.

In the world of the SC-3000 graphics mode, pixel, dot or bit are usually synonyms each other.

Example :

```
10 SCREEN 2,2:CLS
20 FOR A=1 TO 12
30 COLOR A,15
40 FOR I=1 TO 82:PRINTCHR$(144);:NEXT
50 NEXT A
60 FOR C=1 TO 15
70 FOR Y=0 TO 191 STEP 2
80 COLOR ,C,(0,Y)-(255,Y)
90 NEXT Y
100 FOR X=0 TO 255 STEP 3
110 COLOR ,C,(X,0)-(X,191)
120 NEXT X,C
130 GOTO 60
```

Statement	COMSET

Function: Sets data length, controls parity bit

Format: COMSET data length, parity

where

Data length:	5, 6, 7, 8
Parity:	E Even parity
	O Odd parity
	N No parity

At BASIC startup or system boot, the arguments are initialized to be:

Data length = 8
Parity = E

Example:

```
COMSET 8,E
```

Statement	CONSOLE

Function: Sets the cursor scroll limit for the text window, controls the on/off of the click sound, switching between upper and lower cases for characters, and selects printer (#1, #2)

Format: CONSOLE u, l, c, s, p

where

- u Scroll upper limit (0 thru 22)
- l Scroll length (greater than or equal to 2)
- c Click sound on/off (0 = off, 1 = on)
- s Change case (0 = upper case, 1 = lower case)
- p Select printer (1 = printer #1, 2 = printer #2)

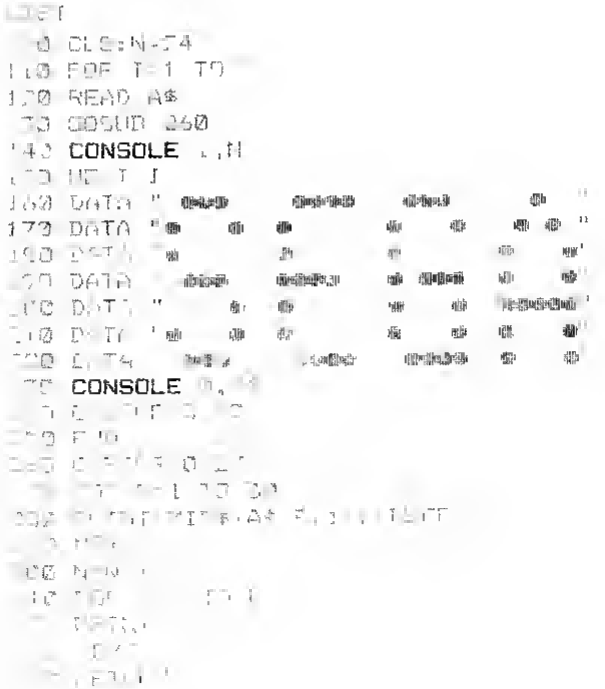
At boot time, each value is initialized as u = 0, l = 24, c = 1, s = 0, p = 1

Description: The values set by this statement are not altered (including program abort) unless delivered by the reset key or reset by another CONSOLE statement

Printer #1 corresponds to the SEGA SP-400

Printer #2 is for a Centronics type printer.

Example:



Statement	CSAVEM

Function: Saves machine language programs onto cassette

Format: CSAVEM "filename", start address, end address

Description: This statement saves the machine language program in memory onto cassette tapes. Filename in this case is limited to up to 16 characters and has no extension

Example:

```
CSAVEM "HEX DATA ",&HF000,&HFFFF
* Saving start
* Saving end
Ready
```

Statement	CURSOR

Function: Sets the cursor on the specified position

Format: CURSOR horizontal position, vertical position

Description: When used on the text window

horizontal position must be in the range: 0 thru 37

vertical position must be in the range: 0 thru 23

When used on the graphics window

horizontal position must be in the range: 0 thru 255

vertical position must be in the range: 0 thru 191

In either of the above cases, ranges out of the ones as specified will cause "Statement parameter error."

If you change the origin of a coordinate on the graphics window with the POSITION statement, the range of the values which can be handled on the coordinate will also change.

The positive range (with respect to the origin) will now be bounded by maximum value−specified coordinate , while the negative range by the negative value of the origin.

The range in this case of course means integer range.

Example :

```
CURSOR 18,12 :PRINT "A"
```

```
10 SCREEN 2,2:CLS
20 CURSOR 125,95:PRINT "A"
```

See also: POSITION

Statement	DATA

Function Supplied data to a READ statement

Format DATA numeric value or character string

Description Multiple number of data can be supplied to this statement as in DATA 1, 2, 3, 4 where comma is used to distinguish each datum.
Character strings need not be enclosed in double quotes except

```
( ) , (,) , (")
```

which must be double-quoted as shown below.

```
" : " , " , " , """"
```

If a numeric datum corresponds to a character string variable in the corresponding READ statement, the datum will be regarded as a character string and hence cannot be used in a numeric expression.
The number of data in the statement and the number of arguments in the corresponding READ statement must be the same.
If the number of data in the statement and the number of arguments in the corresponding READ statement, only the data corresponding to the arguments will be utilized. An error will occur if the number of arguments in a READ statement exceeds that of the data in the corresponding DATA statement.
The READ statement, once executed, reads data from the corresponding DATA statement independent of the latter statement's position in the program.

Example

```
LIST

10 READ A,B,C,D
20 PRINT A+B+C+D
100 DATA 1,2,3,4

RUN
 10
Ready
```

See also READ, RESTORE

Statement	DEF FN

Function Defines user functions

Format **DEF FN** function name (argument) = function definition expression

Description Function name must be longer than 2 characters including the head "FN "
The third character of any function name must be alphabetic, and no reserved word (such as command names) must appear in it.

(Correct) FNA FNB FNCD
(Wrong) FNABS FN1 FMC

Function names are distinguished only by up to 2 characters following "FN "
This means two function names with the same two characters after "FN" are indistinguishable
For example, the following two function names

FNSEGA and FNSE

are regarded to be the same.
Also, the value of the argument you supply to your function does not change after the function invocation.

Example $\sinh x = \frac{e^x - e^{-x}}{2}$, $\cosh x = \frac{e^x + e^{-x}}{2}$

Let's define the above functions

```
10 DEF FNSH(X)=(EXP(X)-EXP(-X))/2
20 DEF FNCH(X)=(EXP(X)+EXP(-X))/2
30 INPUT "X=";X
40 PRINT "sinh(x)=";FNSH(X)
50 PRINT "cosh(x)=";FNCH(X)
```

Statement	DIM

Function: Declares arrays. Dimension is limited up to 3

Format: DIM arrayname (subscript range)
DIM A (20)
DIM BS (5,5), DIM C (2, 3, 4)

Description: Arrays are either one-dimensional array or multi-dimensional. Multi-dimensional array is limited up to 3-dimensional array.
Declaring the one-dimensional array

A (5)

where the number in the parentheses is called a subcript, is equivalent to declaring the following six variables

A (0), A (1), A (2), A (3), A (4), A (5)

Character string arrays can be declared also.
You can use an array element without the necessary declaration but in that case the subscript range is 10.
A two-dimensional array

B (5,5)

and a three-dimensional array

C (3,3,3)

Example:

```
LIST

10 CLS
20 DIM A(9,9)
30 FOR J=1 TO 9
40 FOR K=1 TO 9
50 A(J,K)=J*K
60 IF J*K<10 THEN PRINT" ";
70 PRINT A(J,K);
80 NEXT K
90 PRINT
100 NEXT J
```

See also: ERASE

Statement	DSKI$	(disk input $)

Function: Reads physically from disk into variable

Format: DSKI$ truck number, sector number, character variable, offset, length
Ex: DSKI$ 10, 3, A$(1), 0, 128, A$(2), 128, 128

Description: This statement performs a physical read from the specified sector independent of normal disk operations.

Arguments to this statement are explained as follows:

Truck number: 0 thru 39

Sector number: 1 thru 16

Offset: Specifies from which byte in one sector (256 bytes) the read is to begin.

Length: Specifies how many bytes are to be read from the given offset

One sector consists of 256 bytes. Since a *character string* variable can contain only up to 255 characters, two variables are required to read one whole sector as shown in the following example:

DSKI$ 10, 3, A$(1), 0, 128, A$(2), 128, 128

Where the content of the first half (0 thru 127) of the 3rd sector on the 10th truck is read into A$(1), and then the latter half (128 thru 255) of the same sector is read into A$(2).

See also: DSKO$

Example .

```
10 REM
20 REM   disk sector read
30 REM
40 PRINT
50 INPUT "TRACK  NO.: ;"TR$
60 IF TR$="E" THEN END
70 INPUT "SECTOR NO.: ;"SC
80 TR=VAL(TR$)
90 IF TR 0 OR TR 39 THEN 40
100 IF SC 1 OR SC>16 THEN 40
110 PRINT
120 DSKI$ TR,SC:A$(0',0,128:A$(1),128,
128
130 FOR K=0 TO 1:FOR I=0 TO 15
140 PRINTRIGHT$("0"+HEX$(I*8+K*128),2)
":";
150 D$="" : FOR J=1 TO 8
160 C$=MID$(A$(K),I*8+J,1)
170 PRINT " "RIGHT$("0"+HEX$(ASC(C$)),
2);
180 IF ASC(C$)<32 THEN C$="."
190 D$=D$+C$
200 NEXT J : PRINT "  "
210 GOSUB 250
220 NEXT I,K
230 GOTO 40
240 REM
250 IF INKEY$="" THEN 280
260 IF INKEY$ "" THEN 260
270 IF INKEY$ " " THEN 270
280 RETURN
```

Statement	DSKO$	(disk output $)

Function Writes physically to disks

Format DSKO$ truck number, sector number; character variable, offset, length
Ex. DSKO$ 25, 8, A$(1), 0, 128, A$(2), 128, 128

Note Since this statement writes physically to disks, you must know the physical locations of your files on disk to avoid their destruction with this statement. Use this statement after you have completely mastered the disk and the file structures.

Description This statement does a physical write to the specified sector, independent of normal disk operations.
Arguments to this statement are explained as follows:

Truck number	0 thru 39
Sector number	1 thru 16
Offset	Specifies from which byte in one sector (256 bytes) the write should take place.
Length	Specifies how many bytes should be written starting from the offset.

Any one sector consists of 256 bytes. Since a character string variable can contain only up to 255 characters, filling one sector completely needs two variables as shown in the following example:

```
DSKO$ 25, 8, A$(1), 0, 128, A$(2), 128, 128
```

Where the content of A$(1) is written to the first half (0 thru 127) of the 8th sector on the 25th truck, and the content of A$(2) is written to the latter half (128 thru 255) of the same sector.

Example:

```
10 REM
20 REM   disk sector write
30 REM
40 PRINT
50 INPUT "TRACK  NO.: ";TR$
60 IF TR$="E" THEN END
70 INPUT "SECTOR NO.: ";SC
80 TR=VAL(TR$)
90 IF TR<0 OR TR>39 THEN 40
100 IF SC<1 OR SC>16 THEN 40
110 INPUT "START ADR.: &H";SA$
120 A$(0)="":A$(1)=""
130 S=VAL("&H"+SA$)
140 S=S-INT(S/32768)*65536
150 FOR A=0 TO 1
160 FOR J=S+A*128 TO S+A*128+127
170 K=J-INT(J/32768)*65536
180 A$(A)=A$(A)+CHR$(PEEK(K))
190 NEXT J,A
200 DSKO$ TR,SC:A$(0),0,128:A$(1),128,
128
210 GOTO 40
```

Statement **END**

Function: Puts an end to programs

Format: END

Description: Append this statement to the end of a program if the flow of the program follows the line number.
But those programs having subroutines at their tail must end somewhere before the last statement. Put an end to them with this statement.

Example :

```
10 GOSUB 100
20 PRINT"  LET BASIC STUDY"
30 END
100 FOR N=0 TO 37
200 PRINT"*"120 NEXT N
130 RETURN
```

Statement	ERASE

Function: Cancels array declarations

Format: ERASE
ERASE arrayname, arrayname

Description: If you omit arrayname, all array declarations will be canceled.
With a program, you cannot declare arrays twice under a same name. But if the program flow forces you to do so, use this statement to cancel the former declaration.

Example:

```
100 ERASE
      .
      .
      .
200 ERASE A,B$
```

See also: DIM

Statement **FOR–NEXT–STEP**

Function Repeats lines inserted between the FOR and the NEXT statements

Format FOR numeric variable = initial value TO final value STEP increment
NEXT numeric variable

Description You can insert between the FOR and the NEXT statements the part of your program you want to repeat many times. When the program reaches to the NEXT statement, the variable gets incremented by the amount you specified just after STEP, and that part of yours between the FOR and the NEXT statements is repeated once more.

When the value of the variable reaches to the final value you specified just after TO, then those statement just after the NEXT statement will begin to execute.

If you omit the STEP increment part, the increment defaults to 1.

Note that the increment must be a negative value to "count down" if the initial value is greater than the final value.

The FOR–NEXT statement can be nested (you can put a FOR–NEXT statement within another FOR–NEXT statement), but in which case you must use distinct variables.

A convenient way is to have the NEXT statement two variables, one for the inner and the other for the outer FOR, but in that case you must put the variable for the inner FOR the first.

The depth of one nest can be up to 8.

In the following cases, statements following the FOR statement is executed only once.

Initial value is smaller than final value and increment is negative.
Initial value is greater than final value and increment is positive.
Initial value is equal to final value.
There is no NEXT statement.

Statement	GET #

Function: Read into variable data from random files

Format: GET #file descriptor, record number, variable, offset, length

Description: There is no FIELD statement with this version of BASIC.
The following example illustrates how to replace it
Example:

```
GET #1, 3; A$, 0, 10; A, 10, 8
```

where multiple data specification is demarcated by (,)
This statement reads into the variable data from the file indicated by the file descriptor. The file must previously be opened with the OPEN statement to set the file descriptor. Arguments to the statement are described as follows:

Record number: Must be in the range 1 thru the maximum value specified in the PUT statement. If you omit this, the next record to the record previously processed by the PUT or GET statement is read.

Variable: Can be numeric, character or array type.

Offset: Specifies from which byte in one record (258 bytes) to begin read. Omitting this defaults to 0.

Length: Specifies how many bytes are to be read starting from the offset
The default is to the maximum value of the record, which is 255 in case the offset is 0.
If the variable is a character string array, the length can be arbitrary, but the length for numeric arrays is fixed to be 8 bytes and any length you supply to this statement will be ignored.
Since numeric data are stored according to an internal cord format in random files, non-numeric data are read as 0 or garbase into numeric variables

Example:

```
GET #1, 3; A$, 0, 10; A, 10, 8
```

Reads form the file with its file descriptor 1. First the first 10 bytes of the 3rd record is read into A$, and then the next 8 bytes starting from the 10th byte is read into A

Note Files must be opened with the OPEN statement prior to the GET statement, and must be closed after the GET statement has completed.

The OPEN statement has a different format for random files than for sequential files as in

OPEN filename, AS # file descriptor

Note no "FOR INPUT" is needed here as does for sequential files.

Example :

```
10 CLS
20 PRINT:PRINT
30 PRINT"  initialize of disk ..1"
40 PRINT"  display of music name ..2"
50 PRINT"  rewrite of music name ..3"
60 PRINT:INPUT"  ----    number ";A
70 ON A GOSUB 340,90,220
80 GOTO 10
90 REM GET#
100 INPUT"cassette number  (1~30) ";NO
110 OPEN "cassette .dta" AS #1
120 GET #1,NO;NA$,0,200
130 CLOSE
140 CLS:PRINT TAB(30);"NO.";NO
150 PRINT"number "," name "
160 PRINT:FOR A=0 TO 9
170 S$=MID$(NA$,20*A+1,10)
180 PRINT A+1,S$
190 NEXT A
200 INPUT "";A$
210 RETURN
220 REM   PUT#
230 S$='                              ";NA$=""
240 INPUT"cassette number  (1~30) ";NO
250 FOR A=0 TO 9
260 PRINT"  No.";A+1
270 INPUT" name : ";N$
280 NA$=NA$+LEFT$(N$+S$,20)
290 NEXT A
300 OPEN "cassette .dta" AS #1
310 PUT #1,NO;NA$,0,200
320 CLOSE
330 RETURN
340 REM initialize of disk
350 CLS
360 PRINT"   wating prease "
370 A$=""
380 FOR A=1 TO 20
390 A$=A$+".........."
400 NEXT A
410 OPEN 'cassette .dta" AS #1
420 FOR A=1 TO 30
430 PUT #1,A;A$
440 NEXT A
450 CLOSE
460 RETURN
```

See also PUT, OPEN, CLOSE

Statement	GOSUB – RETURN

Function: Calls and executes a subroutine; after subroutine execution returns to the line succeeding the GOSUB statement.

Format: GOSUB line-number

RETURN

Description: Line-number specifies the first line number of the subroutine. The subroutine is an independent program placed inside or at the end of the program and is called when necessary. Specify a RETURN statement specifies returning to the line succeeding the GOSUB statement.

Control can transfer from a subroutine to another subroutine in a nested subroutine structure.

Subroutines can be nested up to level 8; if this is exceeded, a GOSUB nesting error occurs.

Note: The control returned by a RETURN statement must not go to a RETURN statement. If a RETURN statement is encountered by a statement other than a GOSUB statement, a RETURN without GOSUB error occurs.

Example:

```
10 INPUT"
20 IF
30 IF A
40 GOTO 10
50
60 RETURN
70
80 RETURN
```

See Also: ON GOSUB

Statement	GOTO

Function: Jumps to the specified line number.

Format: GOTO line-number

Description: Program execution starts from the smallest line number. When a GOTO statement is encountered, the control unconditionally jumps to the specified line number.

A direct command can specify starting program execution from an arbitrary line number specified in a GOTO statement. In this case, the variable value remains unchanged. The variable value can be known by directly executing a PRINT variable.

When a RUN or RUN line-number is executed, all variable values are cleared.

Example:

```
10 INPUT"A=";A
20 INPUT"B=";B
30 C=A+B
40 PRINT"A+B=";C
50 GOTO 10
```

See Also: ON GOTO

Statement	HCOPY	(hard copy)

Function Outputs to the printer current screen image

Format HCOPY
HCOPY *n*, enlargement

Description This statement lets you printout current images of the text window or the graphics window.
The function of this statement is governed by the type of your printer.

The SEGA printer SP-400

Only the text window can be printed-out. Also the printable characters are restricted to the ASCII codes only and the graphics symbols for the SC-3000 cannot be printed.

The EPSON RP-80II (Centronics type)

Both the text window and the graphics window can be printed-out. Select printer mode according to the following instructions prior to the execution of HCOPY:

- Hit the Z key while keeping down the control key
- Supply 2 to the CONSOLE statement to select the printer

After you have switched to the printer mode #2, select the window as follows.

HCOPY 1 Printout the text window (1 can be omitted)
HCOPY 2 (Graphics window), enlargement

If you omit *n* in your program, the window currently active will be printed-out.
The enlargement is explained as follows:

0: Standard (0 can be omitted)
1: Double the scale of horizontal direction
2: Double the scale of vertical direction
3: Double the scale of both directions

Example

```
HCOPY
```

Statement	IF—THEN

Function: Conditionally jumps to the specified line number or executes the statement(s) following THEN

Format: IF conditional expression THEN line number
IF conditional expression GOTO line number
IF conditional expression THEN statement(s)

Description: If the conditional expression is true, then either the statement placed after THEN, or the statement indicated by the line number supplied after GOTO or THEN is executed
If the condition is false, the line immediately following the IF—THEN statement is executed.
Conditional expressions are usually comparisons or logical operations. A conditional expression takes the value –1 if the condition is true, and 0 otherwise
You can place more than one statement after THEN, in which case those statements are executed only when the condition is true

Example:

```
10 INPUT"score ";A
20 IF A<50 THEN PRINT"unacceptable"
30 IF A>49 AND A<60 THEN PRINT"borderline"
40 IF A>59 AND A<70 THEN PRINT"acceptable"
50 IF A>69 THEN PRINT"light staff "
60 GOTO 10
```

Statement	INPUT

Function Gets inputs of numeric values and strings of characters from keyboard

Format
INPUT A, B$ numeric or character string variable
INPUT "prompt", numeric or character string variable

Description This statement, once executed in your program, waits for your input by putting a "?" (question mark) onto the screen. If you supply "prompt," then the waiting signal will be "prompt" with no question mark added to it.
Numbers or characters typed in response to the waiting signal (prompt) followed by the CR key will be assigned to the corresponding variables.
If the statement has more than one variable, the waiting signal for the second variable and on will be the string of two consecutive question marks (??).

INPUT A, B, C

Character strings need not be enclosed in double quotes.
The statement displays Redo from start and waits for your input once again if it finds type mismatch between the variable and the data you input.
If you hit just the CR key (without any other characters or numbers) to the statement's input request, following values will be assigned to the variables.

0	when the variable is numeric
null string	when the variable is character string

Null string is the string having no characters in it.

Example :

```
10 CLS
20 CURSOR 10,3:PRINT"menu"
30 CURSOR 10,6:PRINT"1...drink"
40 CURSOR 10,8:PRINT"2...food "
50 CURSOR 10,10:PRINT"3...dessert"
60 CURSOR 10,13:INPUT "order ?";A
70 ON A GOSUB 100,200,300
80 GOTO 60
100 CURSOR 10,16:PRINT"                "
110 CURSOR 10,16:PRINT"coffee...$1.00"
120 RETURN
200 CURSOR 10,16:PRINT"                "
210 CURSOR 10,16:PRINT"cake... $2.00"
220 RETURN
300 CURSOR 10,16:PRINT"                "
310 CURSOR 10,16:PRINT"melon... $300"
320 RETURN
```

Statement	INPUT #

Function: Reads into variables data from indicated files or from RS-232C interface

Format: INPUT #*n*, variable or character string
Where *n* is interpreted to be a file descriptor
if *n* is in the range 1 thru 8
and a port to the RS 232C interface
if *n* is 0

Description: Files must previously be opened with the OPEN statement to get the file descriptor, though you need not use the same descriptor used in a previous PRINT# statement.
If the type of the variable differs from the type of data to be read, no read will take place
Also the order of numeric and character string variables you supply to this statement must agree with that used in the corresponding PRINT# statement, for otherwise the result will be different from what you expect.
Supply the same file name to the OPEN statement as the one you supplied to write data with the PRINT# statement.

Example :

```
10 CLS
20 PRINT "RS-232C TEST"
30 PRINT:PRINT:PRINT
40 PRINT"  1.... SEND    '
50 PRINT'  2.... RECEIVE"
60 PRINT:PRINT:PRINT
70 PRINT'OVER is transmission change'
80 INPUT T
90 ON T GOTO 110,220
100 GOTO 80
110 REM ----SEND   COMENT
120 CLS:PRINT"   SEND LETTER    ·PRINT
130 X=1:Y=2
140 CURSOR X,Y:INPUT A$
150 PRINT #0,A$
160 Y=Y+2:IF Y>=22 THEN CLS:Y=2:PRINT"  SEND LETTER"
170 FOR J=0 TO 20
180 NEXT J
190 IF A$='OVER' THEN 200
200 IF A$="END' THEN END
210 GOTO 140
220 REM ----RECIVE COMENT
230 CLS:PRINT"   RECEIVE LETTER   '
240 X=1:Y=2
250 INPUT #0,B$
260 CURSOR X,Y:PRINT B$
270 Y=Y+2:IF Y >=22 THEN CLS:Y=2:PRINT   RECEIVE LETTER"
280 IF B$="OVER'THEN 120
290 IF B$="END" THEN END
300 GOTO 250
```

See also: PRINT#, OPEN, CLOSE

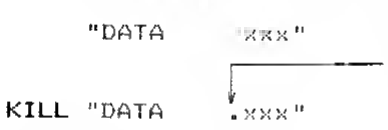
```
10 GOTO 370
20 REM   print #
30 OPEN "birthday.dat" FOR OUTPUT AS #
1
40 GOSUB 260
50 CLOSE:RETURN
60 REM   input #1
70 INPUT"birth month";M:PRINT
80 OPEN "birthday.dat" FOR INPUT AS #1
90 INPUT #1,NA$,YE$,MO,DY,A$
100 IF MO<>M THEN 160
110 PRINT" name  ";NA$
120 PRINT" birthday--> ";YE$;" /";MO;"
/";DY
130 PRINT"blood type-> ";A$
140 INPUT" ";A$
150 PRINT
160 IF EOF(1) THEN 180
170 GOTO 90
180 INPUT " ** data end ** ";A$
190 CLOSE:RETURN
200 REM   append
220 OPEN "birthday.dat" FOR APPEND AS
#1
230 GOSUB 260
240 CLOSE:RETURN
260 PRINT:PRINT
270 INPUT" name     ";NA$
280 INPUT"birth year";YE$
290 INPUT"birth mon.";MO
300 INPUT"birth day ";DY
310 INPUT"blood type";A$
320 PRINT #1,NA$,YE$,MO,DY,A$
330 INPUT" <( E is end )> ";B$
340 IF B$<>"E" THEN 260
350 RETURN
360 REM
370 CLS
380 PRINT"    M E N U "
390 PRINT
400 PRINT"writeing new data....1"
410 PRINT"search of birthday...2"
420 PRINT"in addition to data..3"
430 PRINT:PRINT
440 INPUT"------ push data ";N
450 ON N GOSUB 20,60,200
460 GOTO 370
```

Statement	KILL

Function: Deletes programs saved on disk

Format: KILL "filename"

Description: If a program proved to be no longer useful, you can delete it from disk by specifying its name to this command.
You can do this on the screen after the FILES command.
A program file name includes a delimiter (as in "A FILE xxx"), by a data file does not (as in "DATA xxx"). The KILL command can't delete a file name that does not have a delimiter. To delete such a file, display the file name using the FILES command, move the cursor to this position, type in a delimiter, then execute the KILL command.



If a file is write-protected with the SET command, it can't be deleted with the KILL command.

Statement	LET

Function: Stores (assigns) the right-hand-side value to the left-hand-side variable or an array element

Format: LET variable or an array element = numeric expression
LET character string variable or character string array element = character string

Description: LET is an assignment statement storing the right-hand-side value to the left-hand-side variable or array element
Typing, without "LET"

X = 5

has quite the same effect as typing

LET X = 5

The equal sign '=' above does not mean, as does in mathematics, the equality between the right-hand-side and the left-hand-side

Example:

```
LIST

10 LET A=3
20 LET B=5
30 LET C=A+B
40 PRINT C
50 END

RUN
 8
Ready
```

Statement	LIMIT

Function Sets the end address for the BASIC program area

Format LIMIT end address

Description This statement sets the limit for the BASIC program area, and thereby sets the limit for the user workable area.
You cannot specify an address within the work area for the BASIC interpreter, nor lower than the address as previously set by the NEWON statement.
After the execution of this statement, you can use freely the area higher than or equal to the specified address. The BASIC interpreter will not touch this area.

Example :

```
LIMIT &HF7FF
```

Statement	LINE

Function: Draws line segment connecting specified coordnates.

Format: LINE (X1, Y1) – (X2, Y2) , color code
where
X = horizontal coordinate in the range 0 thru 255
Y = vertical coordinate in the range 0 thru 191

Description: This statement draws the line segment starting from (X1, Y1) and ending at (X2, Y2)
If the origin of the coordinate has been moved by the coordinate to appear, the horizontal distance as well as the vertical distance of these two points must not exceed the range specified above.

Additional function B: Draws a rectangle.

LINE (X1, Y1) – (X2, Y2), color code, B

The above statent draws the rectangle having the line segment connecting (X1, Y1) and (X2, Y2) as its diagonal. What is more, you can paint inside the rectangle by saying.

LINE (X1, Y1) – (X2, Y2), color code, BF

where the color is specified by the color code.
If you omit the starting coordinate (X1, Y1), the draw will begin from the latest point utilized not only by the LINE statement, but the BLILE, PSET or the PRESET statements.

Example :

```
10 SCREEN 2,2:CLS
20 LINE(50,50)-(150,50),1
30 LINE-(50,150),8
```

```
10 SCREEN 2,2:CLS
20 FOR C=0 TO 15
30 LINE(80,50)-(160,100),C,B
40 FOR A=0 TO 300:NEXT A
50 NEXT C
60 GOTO 60
```

See Also: COLOR

Statement	LOADM

Function Loads machine language programs from disk

Format **LOADM** "filename"
LOADM "filename", load start address

Description This statement loads the machine language program on disk indicated by the "filename" in memory.
If you omit the load start address, the load will begin from the address as previously specified by the SAVEM statement. If you specify the load address, the load will begin from the address specified.

Note This statement cannot load programs saved by the SAVE statement.

Example :

```
LIST

10 LIMIT &HCFFF
20 A$="FFFFFFFFFFFFFFFF"
30 PATTERN S#0,A$:PATTERN S#1,A$
40 PATTERN S#2,A$:PATTERN S#3,A$
50 VPOKE &H3B02,0:VPOKE &H3B03,1
60 PRINT" set disk"
70 INPUT" push Y key";Y$
80 IF Y$<>"Y" THEN 70
90 LOADM"example 1.hex",&HD000
100 SCREEN 2,2:CLS:MAG 3
110 CALL &HD000
```

Statement	LPRINT

Function Output to the printer values or character strings

Format

LPRINT	A or A$	Numeric variable or character string variable
LPRINT	A$; B, C	
LPRINT	"X"	Character string
L? A		The "PRINT" can be replaced with "?"

Description This statement is the same with the PRINT statement except the result is written to the printer
"LPRINT" can be abbreviated to "L?".

Note Refer to the manual for your printer before using this statement since there can be a variety of specifications among various printers or from interface to interface

See also PRINT

Statement	MAG	(magnitude)

Function Sets size and magnitude of sprites

Format MAG numeric value

Description Various sizes of sprites can be set by supplying to the MAG statement integers in the range 0 thru 3

MAG 0 Draws 8 by 8-dots' figures in the frame of 8 by 8 picture elements

MAG 1 Draws 16 by 16 dots' figures in the frame of 16 by 16 picture elements by combining 4 patterns of 8 by 8 picture elements (S#0–S#3, S#4–S#8, , S#253–S#255)

MAG 2 Double the size of the pictures drawn by MAG 0 8 by 8 dots' figures will be drawn in the frame of 16 by 16 picture elements, 1 dot becoming equivalent to 2 by 2 picture elements

MAG 3 Double the size of figures drawn by MAG 1 16 by 16-dots' figures will be drawn in the frame of 32 by 32 picture elements by combining 4 patterns of 16 by 16 picture elements 2 by 2 picture element becomes equivalent to 1 dot

Combining 4 patterns to create a figure as in the cases MAG 1 and MAG 3 above can be done with a single SPRITE statement Since sprite names are synonyms for pattern numbers (S#number), you can, for example, let one pattern number among the group S#0–S#3 be a sprite name to automatically construct the S#0–S#3 pattern

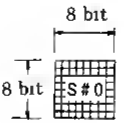
Note In cases MAG 1 and MAG 3 above, the possible combinations of patterns are not arbitrary

If you make some mistake in numbering the patterns, the resulting figures will be different from what you expect

Figure

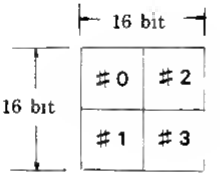
The MAG statement is used to specify the scale of figures drawn by the PATTERN statement. In this figure, one picture element corresponds to one bit.

MAG 0



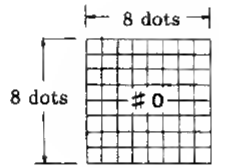
1 bit = 1 dot

MAG 1



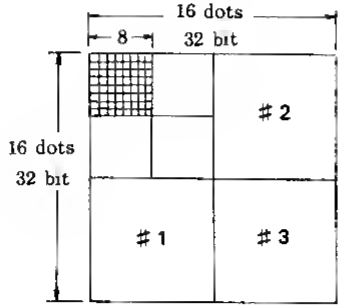
Four of MAG 0 are combined to draw the pattern.

MAG 2



2 bit × 2 bit is deemed as 1 dot.

MAG 3



Patterns are drawn by combining four MAG 2.

Example :

```
10 REM  --- MAG & PATTERN TEST ---
20 SCREEN 2,2:CLS
30 PATTERN S#0,"0103070F1F3F7FFF"
40 PATTERN S#1,"FF00FF00FF00FF00"
50 PATTERN S#2,"80C0E0F0F8FCFEFF"
60 PATTERN S#3,"AAAAAAAAAAAAAAAA"
70 X=32:Y=90:XX=0
80 PRINTCHR$(17)
90 MAG M
100 FOR T=0 TO 3
110 BLINE(0,16)-(255,24),,BF
120 CURSOR 0,0:PRINT"  MAG & PATTERN T
EST"
130 CURSOR 0,16:PRINT"  MAG";M;":PATTE
RN S#";:CURSOR204,16:PRINT T
140 SPRITE 2,(X,Y),T,T+1
150 SPRITE 0,(X+32,Y),T,T+3
160 SPRITE 3,(X+64,Y),T,T+5
170 SPRITE 5,(X+96,Y),T,T+7
180 FOR W=0 TO 130
190 SPRITE 1,(160,W),T,14
200 NEXT W
210 FOR WT=0 TO 100:NEXT WT
220 NEXT T
230 M=M+1:IF M=4 THEN M=0:T=0
240 GOTO 90
```

Statement	NAME (name)

Function: Changes filename of a program on disk

Format: NAME "current filename" AS "new filename"

Description: Use this command to change the file name of a program, where it is understood that file name is equivalent to program name.
You must supply the exact file name including the extension.
Otherwise the command won't know which file you want to rename.
To rename "central-BAS" to "pacific-BAS," type NAME "central-BAS" AS "pacific-BAS."

Example:

```
NAME "OLD NAME.    " AS "NEW NAME.    "
```

Statement	ON GOSUB

Function Jumps to one of the subroutines specified by the line numbers according to the variable

Format **ON** variable **GOSUB** line number, line number, line number

Description Jumps to one of the subroutines indicated by the line numbers specified after GOTO according to the value of the variable previously assigned by a numeric expression or by an **INPUT** statement

The value is an integer and must be taken in the range 1 thru the number of line numbers you specify after **GOTO**, each integer corresponding to each line number

The **RETURN** statement is used on return from subroutines

Example

```
10 CLS
20 CURSOR 10,2:PRINT"menu"
30 CURSOR 10,6:PRINT"1...drink"
40 CURSOR 10,8:PRINT"2...food"
50 CURSOR 10,10:PRINT"3...dessert"
60 CURSOR 10,13:INPUT"order";A
70 ON A GOSUB 100,200,300
80 GOTO 60
100 CURSOR 10,16:PRINT"                "
110 CURSOR 10,16:PRINT"coffee...¥700"
120 RETURN
200 CURSOR 10,16:PRINT"                "
210 CURSOR 10,16:PRINT"cake...  ¥200"
220 RETURN
300 CURSOR 10,16:PRINT"                "
310 CURSOR 10,16:PRINT"melon...  ¥250"
320 RETURN
```

Statement	ON GOTO

Function: Jumps to one of the specified lines according to the variable

Format: ON variable GOTO line number, line number, line number

Description: Jumps to one of the lines specified after GOTO according to the value of the variable previously assigned by a numeric expression or by an INPUT statement. The value is an integer and must be taken in the range 1 thru the number of line numbers you specify after GOTO, each integer corresponding to each line number.

If the value got greater than the number of line numbers, the line immediately following this statement would be executed

Example:

```
10 INPUT"order";A
20 ON A GOTO 100,200,300
30 GOTO 10
100 PRINT "coffee":GOTO 10
200 PRINT "cake":GOTO 10
300 PRINT "milk":GOTO 10

RUN
order ?1
coffee
order ?2
cake
order ?3
milk
order ?
Break in 10
```

Statement	OPEN

Function Opens sequential files

Format OPEN "filename" FOR "mode" AS #file-descriptor
where "mode" must be one of

INPUT	read from disk into memory
OUTPUT	write from memory onto disk
APPEND	append onto disk

and file-descriptor must be one of 1, 2, 3, 4,

Description This statement opens disk for data read/write.
No disk read/write can take place without opening it beforehand
The file descriptor represents the file name during the disk open
The file descriptor can be set to a maximum number of 8 as specified with the MAXFILE command When disk basic is started, the maximum number of files that can be opened is defaulted at 3

Example

```
10 OPEN "DATA      .     " FOR INPUT AS #1
20 INPUT #1,A$
30 CLOSE
```

Statement	OUT

Function Outputs data to specified output port

Format OUT output port number, data

Description Output port number are predetermined by the system for outputting data to external devices

Example:

```
10 SOUND 1,262,0
20 SOUND 2,294,0
30 SOUND 2,330,0
40 FOR A=0 TO 15 STEP .5
50 OUT &H7F,&H90+A:REM
50 OUT &H7F,&H90+A:REM  turn on tone
70 OUT &H7F,&HD0+A:REM
90 NEXT A
90 GOTO 40
```

Statement	PAINT

Function Paints inside or outside areas formed by bits 1

Format PAINT (X, Y), color code

Description Use this statement to paint inside or outside those areas drawn by the LINE or the CIRCLE statement. But note that even a one-bit hole in such regions will cause the color wooze out from the hole.
Make sure lines have no break points on them.
Use the RESET key to interrupt or stop the statement since the painting cannot be interrupted by the BREAK key.

Example

```
10 SCREEN 2,2:CLS
20 FOR I=0 TO 255 STEP 16
30 LINE(I,0)-(I,191):NEXT I
40 FOR I=0 TO 191 STEP 16
50 LINE(0,I)-(255,I):NEXT I
60 C=RND(1)*16
70 X=RND(1)*256:Y=RND(1)*192
80 PAINT(X,Y),C
90 GOTO 60
```

Statement	PATTERN

Function: Sets character or sprite pattern.

Format: To set a character pattern

PATTERN C= character code, numeric character string where character code must be in the range 32 thru 255 to set a sprite pattern.

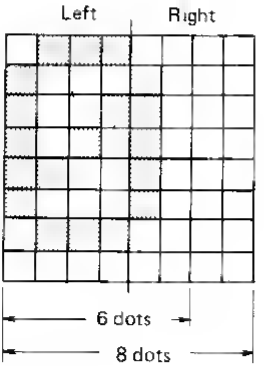
PATTERN S= sprite name, numeric character string

Where sprite name is an integer in the range 0 thru 255 which can also be supplied as a hexadecimal number.

Description: In both of the above formats, the numeric character string must be supplied as a hexadecimal number.

The format for character patterns differs from that of sprite patterns.

Character pattern (characters and symbols that can be input from the keyboard).
The pattern is constructed out of the 8-by-8 dots' square (see figure below).
In this frame, the bottom row and the rightmost column are left blank so that characters do not touch each other vertically and horizontally.
Besides, the rightmost two columns are ignored for character patterns.
So only the first 6 columns and the first 7 rows in the frame are utilized for character patterns.



Binary representation		Hexadecimal representation	
Left	Right	Left	Right
0 1 1 1	0 0 0 0	7	0
1 0 0 0	1 0 0 0	8	8
1 0 0 1	1 0 0 0	9	8
1 0 1 0	1 0 0 0	A	8
1 1 0 0	1 0 0 0	C	8
1 0 0 0	1 0 0 0	8	8
0 1 1 1	0 0 0 0	7	0
0 0 0 0	0 0 0 0	0	0

Shadowed square = bit 1 Blank square = bit 0

```
PATTERN C#92 , "708898A8C8887000" CR
```

Now type

92 in the C#92 above corresponds to the character "¥" the asch code of which is 92.

Now press the "¥" key, and you will see a "0" appear on the screen

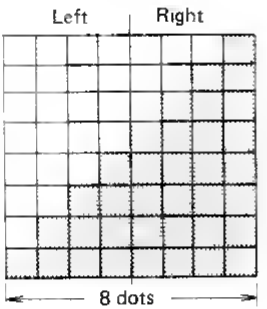
This means the pattern corresponding to the asch code 92 has just been replaced by the one you input with the PATTERN statement.

Since patterns defined in this way remain unchanged until you power-off the computer or re-boot the system, you must be careful not to meddle the ordinary keys with your patterns.

If you do that, talking to your computer such as inputting programs will become much confusing.

Sprite pattern (used only on the graphics window)

Like character patterns, sprite patterns are constructed out of 8-by-8 dots' square. But unlike them, you can use the entire square for the sprite patterns.



0 0 0 0	0 0 0 1	0	1
0 0 0 0	0 0 1 1	0	3
0 0 0 0	0 1 1 1	0	7
0 0 0 0	1 1 1 1	0	F
0 0 0 1	1 1 1 1	1	F
0 0 1 1	1 1 1 1	3	F
0 1 1 1	1 1 1 1	7	F
1 1 1 1	1 1 1 1	F	F

```
PATTERN S#0 "0103070F1F3F7FFF"
```

Note The PATTERN statement uses different formats for character patterns and for sprite patterns.

C# for character patterns

S# for sprite patterns

See Also SPRITE, MAG

Get a sheet of graph section paper and draw an 8-by-8 square on it
Now shadow appropriate squares in the frame to realize your image of the pattern you want
Write sequences of 0's and 1's beside each row in the frame following the rule

a shadowed square corresponds to 1

a blank square corresponds to 0

In this way you get 8 rows of binary numbers, each binary number corresponding to each row in the frame
Now divide each binary number in two from the center to get two binary numbers having 4 places
You have now 8 rows of 2 binary numbers
Translate them into hexadecimal using the conversion table given below

For example, the row

becomes

0 1 1 1 0 0 0 0

the left half of which is 7 in hexadecimal and the right half 0 yeilding "70"
Supply the hexadecimal number thus got to the PATTERN statement and you will see, by pressing an appropriate key, your pattern displayed on the screen
Use 8 by 8 square for graphics patterns, and 8 by 6 sequare for character patterns

Conversion Table

10 decimal	2 binary	16 hexadecimal
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0 Shift	2
3	0 0 1 1	3
3	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10 Shift	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F
16	1 0 0 0 0	10 Shift

```
10 PATTERN C#48,"3048484848483000"
20 PATTERN C#49,"2060202020207000"
30 PATTERN C#50,"7088881020408800"
40 A$(0)="20A8702070A82000"
50 A$(1)="0000000000000000"
60 A$(2)=A$(1)
70 CLS
80 FOR B=9 TO 3 STEP -3
90 X1=19-B*2:X2=19+B*2
100 Y1=11-B:Y2=11+B
110 C=0:FOR L=X1 TO X2
120 Y=Y1:X=L:GOSUB 260
130 Y=Y2:X=X2-L+X1:GOSUB 260
140 C=C+1:NEXT L
150 C=0:FOR L=Y1 TO Y2
160 X=X1:Y=Y2-L+Y1:GOSUB 260
170 X=X2:Y=L:GOSUB 260
180 C=C+1:NEXT L
190 NEXT B
200 FOR K=0 TO 3:C=0:FOR L=0 TO 3
210 PATTERN C#48,A$(CMOD3)
220 PATTERN C#49,A$((C+2)MOD3)
230 PATTERN C#50,A$((C+1)MOD3)
240 C=C+1:NEXT L,K
250 GOTO 200
260 VPOKE &H3C00+X+Y*40,CMOD3+48
270 RETURN
```

Statement	POKE

Function: Writes data to memory

Format: POKE address, data

Description: This statement writes one byte datum to the specified address in memory.
The range of the address into which data can be written with this statement varies according to the state of current programs in memory.
Note that if you write into the **BASIC** program area, the program may go wild. Use this statement after you have carefully analyzed the program area to make sure you are writing to a safe place.

Example:

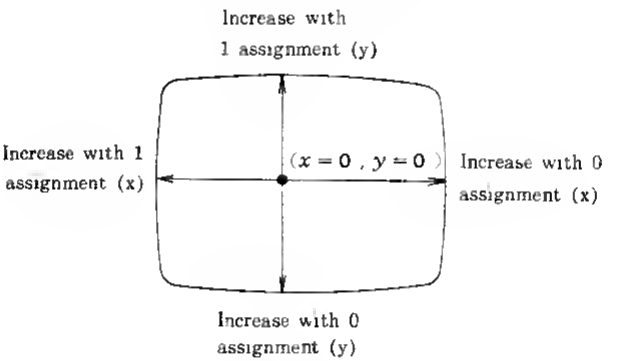
```
10 LIMIT &HFEFF
20 FOR A=&HFF00 TO &HFF0E
30 READ D$
40 POKE A,VAL("&H"+D$)
50 NEXT A
60 PRINT "BOOT OK ?(Y OR N)"
70 IF INKEY$="" THEN 70
80 IF INKEY$<>"Y" THEN END
90 CALL &HFF00
100 DATA 3E,0C,D3,E7,C3,00,00
```

Statement	POSITION

Function Sets the origin of a coordinate for the graphics window

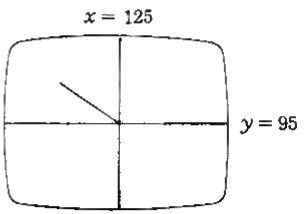
Format POSITION (x, y), x-axis' direction, y-axis' direction
where
x = integer in the range 0 thru 255
y = integer in the range 0 thru 191
x-axis' direction 0 = rightward increase 1 = leftward increase
y-axis' direction 0 = downward increase 1 = upward increase

Description The origin is set as POSITION (0,0),0,0 immediately after the RUN, meaning the origin is located on the topmost, extreme left.
This statement takes effect only on the graphics window



Example

```
10 SCREEN 2,2:CLS
20 POSITION(125,95),0,1
30 FOR N=0 TO 30
40 PSET(X,Y),1
50 X=X+1:Y=Y+1
60 NEXT N
```




```
10 SCREEN 2,2:CLS
20 POSITION(100,50),0,0
30 FOR N=-10 TO 1 STEP .1
40 X=N*20+120:Y=SIN(N)*50+45
50 PSET(X,Y),1
60 NEXT N
```

See also CURSOR

Statement	PRESET

Function Erases dot on the specified coordinate.

Format **PRESET** (X, Y)

Description The statement erases the dot (a pixel) on the specified coordinate (X, Y).
The color cannot be specified to this statement, and the color corresponding to bit "0" is chosen to erase the dot.

Example

```
10 SCREEN 2,2:CLS
20 PAINT(0,0),8
30 X=RND(1)*255:Y=RND(1)*191
40 PRESET(X,Y)
50 GOTO 30
```

See Also PSET, COLOR

Statement	PRINT

Function Displays on the TV screen values of numeric variables, character variables, or character strings

Format

PRINT A or AS	Numeric variable or character string variable
PRINT AS B C	
PRINT "X"	Character string
?A	

Description You use this statement to display on the screen results of calculations or strings of characters

This statement is a must to computer usage and so do never forget it. For otherwise there will be no display on your TV screen.

Some special characters used with this statement have special meaning to it and you must remember them.

Character strings must be enclosed in double quotes (" ") as in

PRINT "A B C"

Variables can be demarcated either by ' ; ' (semicolon) or ' , ' (comma) with the following effects

PRINT A;B	Displays as A B (one after the other)
PRINT A,B	Displays as A ← (18 spaces) → B

PRINT adds one space at the head of each numeric values, for it needs the space for '+' or '–' sign. The '+' sign does not appear on the screen, however.

Example :

PRINT CHR$(12)

Statement	PRINT #

Function Outputs values or character strings to specified sequential files or RS-232C interface

Format PRINT #*n*, variable, or character string
Where *n* is interpreted to be a file descriptor
if *n* is in the range 1 thru 8,
and a port to the RS-232C interface
if *n* is 0

Description Files must previously be opened with the OPEN statement to get the file descriptor.
Numeric variables and character string variables can be demarcated by ',' to write them out successively as in:

```
PRINT #1, A$, A; B$, B
```

Note The usual comma ',' used in PRINT# statement in usual BASICs cannot be used in the Disk Basic for the SC-3000
Files must be closed after the processing with this statement has completed

Example:

```
10 CLS
20 PRINT "RS-232C TEST"
30 PRINT:PRINT:PRINT
40 PRINT"  1.... SEND    "
50 PRINT"  2.... RECEIVE"
60 PRINT:PRINT:PRINT
70 PRINT"OVER is transmission change"
80 INPUT T
90 ON T GOTO 110,220
100 GOTO 80
110 REM ----SEND    COMENT
120 CLS:PRINT"   SEND LETTER  ":PRINT
130 X=1:Y=2
140 CURSOR X,Y:INPUT A$
150 PRINT #0,A$
160 Y=Y+2:IF Y>=22 THEN CLS:Y=2:PRINT"
  SEND LETTER"
170 FOR J=0 TO 20
180 NEXT J
190 IF A$="OVER" THEN 220
200 IF A$="END" THEN END
210 GOTO 140
220 REM ----RECEIVE COMENT
230 CLS:PRINT"   RECEIVE LETTER  "
240 Y=4:Y=2
250 INPUT #0,S$
260 CURSOR X,Y:PRINT S$
270 Y=Y+2:IF Y >=22 THEN CLS:Y=2:PRINT"
  RECEIVE LETTER"
280 IF S$="OVER" THEN 120
290 IF S$="END" THEN END
300 GOTO 250
```

See also INPUT#, OPEN, CLOSE

```
10 GOTO 270
20 REM   print #
30 OPEN "birthday.dat" FOR OUTPUT AS #1
40 GOSUB 260
50 CLOSE:RETURN
60 REM   input #1
70 INPUT"birth month";M:PRINT
80 OPEN "birthday.dat" FOR INPUT AS #1
90 INPUT #1,NA$,YE$,MO,DY,A$
100 IF MO<>M THEN 160
110 PRINT" name  ";NA$
120 PRINT" birthday--> ";YE$;" /";MO;"/";DY
130 PRINT"blood type-> ";A$
140 INPUT" ";A$
150 PRINT
160 IF EOF(1) THEN 180
170 GOTO 90
180 INPUT " ** data end ** ";A$
190 CLOSE:RETURN
200 REM   append
210 OPEN "birthday.dat" FOR APPEND AS #1
220 GOSUB 260
230 CLOSE:RETURN
260 PRINT:PRINT
270 INPUT" name      ";NA$
280 INPUT"birth year";YE$
290 INPUT"birth mon.";MO
300 INPUT"birth day ";DY
310 INPUT"blood type";A$
320 PRINT #1,NA$,YE$,MO,DY,A$
330 INPUT" <<  E is end >> ";B$
340 IF B$<>"E" THEN 260
350 RETURN
360 REM
370 CLS
380 PRINT"    M E N U "
390 PRINT
400 PRINT"writeing new data....1"
410 PRINT"search of birthday...2"
420 PRINT"in addition to data..3"
430 PRINT:PRINT
440 INPUT" <---  push data ";N
450 ON N GOSUB 20,60,200
460 GOTO 370
```

Statement	PSET

Function Puts dot on the specified coordinate

Format **PSET** (X, Y), color code

Description The statement puts dot (a pixel) on the coordinate (X, Y) on the screen. If you omit color code, the color used by a previous statement will be used

Example

```
10 SCREEN 2,2:CLS
20 X=0:Y=95:E=1
30 PSET(X,Y),8
40 X=X+1:Y=Y+E
50 IF Y=120 THEN E=-1
60 IF Y=85 THEN E=1
70 IF X=250 THEN END
80 GOTO 30
```

See Also PRESET, COLOR

Statement	PUT #

Function: Writes data or values of expression to random files

Format: PUT #file descriptor, record number, variable, offset, length

Description: There is no FIELD statement in this BASIC. The following example illustrates how to replace it.

Example:

```
GET #1, 3, A$, 0, 10, A, 10
```

This statement writes contents of the variables to the file indicated by the file descriptor previously returned by an OPEN statement.

The arguments to this statement are described as follows:

Record number: Must be an integer in the range 1 thru the maximum record number previously written by PUT statement. If you omit this, the next record to the record processed by preceeding PUT or GET will be written.

Variable: Can be numeric, character string or array.

Offset: Specifies from which byte of one record (256 bytes) the processing should start.
Omitting this defaults to 0.

Length: Specifies how many bytes are to be written from the offset.
If you omit this, the length of the content of the variable will be used.
If you specify a length and the actual length falls short of the specified length, the remaining bytes are filled with 00H.
On the contrary, if the actual length exceeds the length you specify to the statement, the extra bytes are discarded.

Example:

```
PUT #1, 3, A$, 0, 10, A, 10, 8
```

This statement writes to the file indicated by the file descriptor 1.
First the content of A$ which is 10-bytes long is written from the beginning of the 3rd record, and then the content of A which is 8-bytes long is written from the 10th byte of the same record.

Note: Files must be opened with the OPEN statement prior to the execution of the PUT statement, and must be closed with the CLOSE statement after the processing has completed.

The OPEN statement uses a different format than in the case for sequential files as in

OPEN filename, AS #file descriptor

Note there needs no "FOR OUTPUT" as does in sequential files

Example:

```
10 CLS
20 PRINT:PRINT
30 PRINT"  initialize of disk...1"
40 PRINT"  display of cassette ...2"
50 PRINT"  rewrite of disk  ......3"
60 PRINT:INPUT"  ----   number ";A
70 ON A GOSUB 230,90,220
80 GOTO 10
90 REM GET#
100 INPUT"cassette number (1~30)  ";NO
110 OPEN "cassette  .dta" AS #1
120 GET #1,NO;NA$,0,200
130 CLOSE
140 CLS:PRINT TAB(30);"NO.";NO
150 PRINT"number ","  name "
160 PRINT:FOR A=0 TO 9
170 S$=MID$(NA$,20*A+1,10)
180 PRINT A+1,S$
190 NEXT A
200 INPUT "";A$
210 RETURN
220 REM   PUT#
230 S$='                              ":NA$=""
240 INPUT"cassettenumber (1~30)  ";NO
250 FOR A=0 TO 9
260 PRINT"  No.";A+1
270 INPUT" name : "zu$
280 NA$=NA$+LEFT$(N$+S$,20)
290 NEXT A
300 OPEN "cassette  .dta" AS #1
310 PUT #1,NO;NA$,0,200
320 CLOSE
330 RETURN
340 REM initialize of disk
350 CLS
360 PRINT"   wating prlase   "
370 A$=""
380 FOR A=1 TO 20
390 A$=A$+".........."
400 NEXT A
410 OPEN "cassette    .dta" AS #1
420 FOR A=1 TO 30
430 PUT #1,A;A$
440 NEXT A
450 CLOSE
460 RETURN
```

See also: GET, OPEN, CLOSE

Statement	READ

Function Reads data specified by a DATA statement

Format READ variable name or array name
READ A or READ A, B, C$

Description This statement must be paired with a DATA statement
The READ statement reads the data supplied to a DATA statement placed anywhere in the program.
The variables to a READ statement can either be numeric or string, but if the type of a datum to be read differs from that implied by the variable, the mismatch error will occur.
The READ statement can take a multiple number of arguments as in

```
READ A, A$, B, B$
```

but the number of arguments in the statement must agree with the number of data in the corresponding DATA statement.

```
READ A, A$, B, B$
       ↓   ↓   ↓   ↓        types of variables
DATA 10, apple, 5, orange
```

If the number of arguments to a READ statement exceeds that of the data in the corresponding DATA statement, an error will occur. If, on the contrary, the number of data in a DATA statement exceeds that of arguments in the corresponding READ statement, the remaining data will either be ignored or read by the next READ statement.
In case there are more than one DATA statement in a program, the READ statement is used to read them all.

Example :

```
LIST

10 READ A,B,C,D
20 PRINT A+B+C+D
100 DATA 1,2,3,4

RUN
 10
Ready
```

See also DATA, RESTORE

Command	REM	(remarks)

Function: Marks comment

Format: REM

Description: Use this statement to insert remarks in your program.
The BASIC interpreter will ignore the lines beginning with REM.

Example:

```
10 REM* CALCULATOR *
20 CLS
30 PRINT 2+3
```

Statement	RESTORE

Function Specifies a DATA statement to be read by the next READ statement

Format **RESTORE** line number

Description In a program with more than one DATA statement this statement is used to declare that the DATA statement associated with the given line number is to be read next

If you omit line number, the next instance of a READ statement will read from the first DATA statement in the program

To read the same data repeatedly, place this statement before the READ statement

If you supply "line number," the DATA statement specified by the number will be read independent of its location in the program

Example

```
LIST

10 READ A,B,C,D
20 DATA 1,2,3,4
30 RESTORE
40 READ E
50 PRINT A+B+C+D+E

RUN
 11
Ready
```

Statement	SAVEM

Function: Saves machine language programs onto disk

Format: SAVEM "filename", start address, end address

Description: This statement saves the machine language program in memory onto disk in a file named "filename".
Filename must accord with those of program files for disks.

"filename . extension"
↑ ↑
max. 8 characters max. 3 characters

Note: The LOAD statement will not load the file saved by SAVEM.

Example:

```
LIST

10 LIMIT &HCFFF
20 FOR AD=&HD000 TO &HD07B
30 READ D$:D=VAL("&H"+D$)
40 POKE AD,D
50 NEXT AD
60 INPUT"set disk and push Y key";Y$
70 IF Y$<>"Y" THEN 60
80 SAVEM"example 1.hex",&HD000,&HD07B
90 END
100 REM   machine data
110 DATA C5,D5,E5,F5,11,01,01,01
120 DATA 64,64,7A,80,21,A8,10,CD
130 DATA 60,D0,30,06,7A,ED,44,57
140 DATA 18,F1,47,7B,81,21,D0,10
150 DATA CD,60,D0,30,06,7B,ED,44
160 DATA 5F,18,F1,4F,CD,37,D0,00
170 DATA 00,CD,71,D0,C3,0A,D0,F5
180 DATA E5,21,00,3B,F3,CD,4C,D0
190 DATA 78,CD,59,D0,79,CD,59,D0
200 DATA FB,E1,F1,C9,F5,7D,D3,BF
210 DATA 7C,E6,3F,F6,40,D3,BF,F1
220 DATA C9,00,00,00,00,D3,BE,C9
230 DATA F5,BC,38,0A,E5,67,7D,BC
240 DATA E1,38,03,F1,A7,C9,F1,37
250 DATA C9,C5,0E,FF,0D,D9,D9,20
260 DATA FB,00,C1,C9
```

See Also: LOADM, LIMIT

Statement	SCREEN

Function: Controls the active and the visual windows

Format: SCREEN active window, visual window

Description: SC-3000 has two independent windows.

1: Text window for program input

2: Graphics window for graphics display

The BASIC interpreter initializes both of the windows to 1.

SCREEN 1, 1

You must execute, prior to any graphics commands:

SCREEN 2, 2

The active window is utilized by the PRINT statement and so on, while the visual window is for graphics output.

The CLS statement erases the active window implied by the SCREEN statement.

Example :

```
SCREEN 2,2:CLS
Ready
```

Statement	SET

Function Sets file mode

Format SET "filename" , "P" (read-only)

Description The read only mode of a file can be reset by supplying the file name and any other character than "P" after the ' , ' to this command
Once you set the read-only mode with this command, any modified version of the program cannot be saved under the same file name Save it under other file name
A file set to read-only by this command cannot be deleted even with the KILL command

Example

```
SET "SAMPLE  ","R"
```

Statement	SOUND

Function Generates sounds having given frequencies

Format SOUND channel, frequency, volume

Usage SOUND 1, 1000, 15 CR
Ready

Description 〈Channel〉
Each channel corresponds to a certain fixed tune.
By mixing the first three channels, you can play a trio.

Channel	Function
0	Turn off the sound
1	Generate ordinary notes
2	Generate ordinary notes
3	Generate ordinary notes control frequency when the channel specified is 4 or 5
4	Generate white noises
5	Generate synchronized noises

〈Frequency〉
Specify desired frequency if the channel selected is 1, 2 or 3.
If the selected channel is 4 or 5, specify one of the integer among 0 thru 3 according to the following description.

0 thru 2: Each corresponds to a predetermined frequency
3: Frequency is controlled by the channel 3

〈Volume〉

0: Switch off the sound
1: Minimum volume
⋮
15: Maximum volume

With this statement you can produce amusing sound effects to your games or compose and produce melodies. See the following table.

Example

```
LIST

10 RESTORE 80
20 READ D
30 IF D=0 THEN SOUND0:END
40 SOUND 1,D,15
50 SOUND 2,D*2,11
60 SOUND 3,D*3,9
70 GOTO 20
80 DATA 370,370,392
81 DATA 440,440,392
82 DATA 370,330,294
83 DATA 294,330,374
84 DATA 370,330,330
85 DATA 370,370,392
86 DATA 440,440,392
87 DATA 370,330,294
88 DATA 294,330,370
89 DATA 330,294,294,0
```

This program makes use of synchronized noises.
The channel 3 controls frequency while the channel 5 controls volume

```
LIST

10 FOR I=1500 TO 3000 STEP 10
20 SOUND 3,I,0
30 SOUND 5,3,15-ABS(I/100-20)
40 NEXT I
50 SOUND0
```

ORGAN

```
LIST

10 REM ----- doremi ---
20 CLS
30 PRINT" #la   #do re   #fa so la   #do re
"
40 PRINT"  W     R  T     U  I  O     @  [ "
50 PRINT
60 PRINT" A  S  D  F  G  H  J  K  L  ;  :  ]"
70 PRINT" la ti do re mi fa so la ti do re mi
"
80 Z$=INKEY$
```

```
90 IF Z$="A" THEN SOUND1,220,15
100 IF Z$="W" THEN SOUND1,233,15
110 IF Z$="S" THEN SOUND1,247,15
120 IF Z$="D" THEN SOUND1,262,15
130 IF Z$="R" THEN SOUND1,277,15
140 IF Z$="F" THEN SOUND1,294,15
150 IF Z$="T" THEN SOUND1,311,15
160 IF Z$="G" THEN SOUND1,330,15
170 IF Z$="H" THEN SOUND1,349,15
180 IF Z$="U" THEN SOUND1,370,15
190 IF Z$="J" THEN SOUND1,392,15
200 IF Z$="I" THEN SOUND1,415,15
210 IF Z$="K" THEN SOUND1,440,15
220 IF Z$="O" THEN SOUND1,466,15
230 IF Z$="L" THEN SOUND1,494,15
240 IF Z$=";" THEN SOUND1,523,15
250 IF Z$="@" THEN SOUND1,554,15
260 IF Z$=":" THEN SOUND1,587,15
270 IF Z$="[" THEN SOUND1,622,15
280 IF Z$="]" THEN SOUND1,659,15
290 IF Z$="" THEN SOUND0
300 GOTO 80
```

Frequency Table

Notes			f1	f2	f3	f4	f5
C	do	f		131	262	523	1047
C#, D♭				139	277	554	1109
D	re	g		147	294	587	1175
D#, E♭				156	311	622	1245
E	mi	a		165	330	659	1319
F	fa	b		175	349	698	1397
F#, G♭				185	370	740	1480
G	so	c		196	392	784	1568
G#, A♭				208	415	831	1661
A	la	d	110	220	440	880	1760
A#, B♭			117	233	466	932	
B	si	e	123	247	494	988	

Unit: Hz

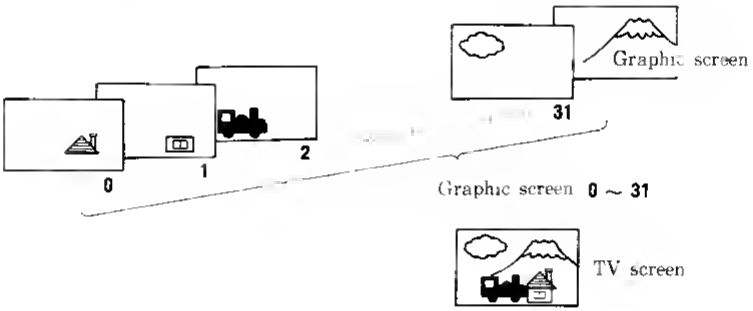
Statement	SPRITE

Function Moves sprite patterns on the screen

Format SPRITE sprite window, (X, Y), sprite name, color code

Description This statement is used to move figures constructed by the PATTERN statement to the specified coordinate on the screen to construct a sprite pattern.
No re-definition of patterns is needed.
The arguments to the statement are as follows.

Sprite window.
An integer in the range 0 thru 32 each corresponding to one sprite window.
A window with a lower id number is placed in front of a window with a higher id number.

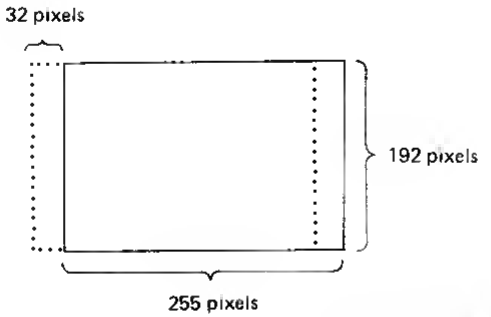


Sprite name.
This is the number you supplied to the PATTERN statement.
A sprite name can be used on more than one sprite windows.
The number of sprite patterns can be up to 56 and 32 of them can be simultaneously displayed onto the screen.

Color code.
One sprite has one color.

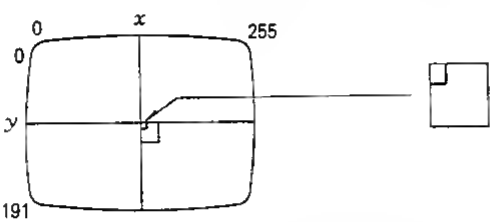
Coordinate.
If you move sprites rightward with the horizontal range exceeding 255, they will reappear from the left border of the screen, and this is the intended result.
Remind this fact when moving figures horizontally.

In case a sprite moves leftward beyond the left margin of the screen, the sprite window is shifted to left by the amount of 32 pixels and the sprite is entirely erased from the screen
In this case you are recommended to set a left margin in your coordinate since continuing the move will cause the "Parameter error".



Note: The dotted line indicates the sprite window shifted by the EC (Early Clock) bit.

The origin of the 8-by-8 frame regarded as a coordinate for character patterns is on the uppermost, extreme left, which is also the case for sprite windows.



The SPRITE statement has the nice characteristic of being able to put figures in front of or on the back of others by drawing them on different sprite windows.
A sense of perspective can easily be introduced into your graphics pictures by utilizing this characteristics intelligently

Note 1: Although you can display up to 32 sprite windows simultaneously, only up to 4 windows can be placed on the same horizontal line (rastor) and the 5th window, if supplied, becomes invisible blocked by the former windows. But since windows are blocked not by sprites, but by dots, if you move the 5th window vertically, the window will begin to be blocked and reappear dot-wise on the screen.

Note 2: Although each sprite window can have one color, you can combine from 2 to 4 windows to create a, say, 4-colored character
In case you must put multiple number of sprite windows on a same horizontal line, plan them carefully remindng the note 1 above

Example:

```
10 M=1
20 SCREEN 2,2:CLS
30 MAG M:C=RND(1)*15+1
40 CURSOR 10,10:PRINT CHR$(17);"MAG";M
50 FOR Y=0 TO 191 STEP 4
60 PATTERN S#0,"00193F3C1C0D0F7B"
70 PATTERN S#1,"0C0F0F0F07031B07"
80 PATTERN S#2,"00CCFE9E9CD878EC"
90 PATTERN S#3,"1AFAF8F0EC7C3800"
100 Y1=Y:GOSUB 190
110 PATTERN S#0,"00193F3C1C0D0F1B"
120 PATTERN S#1,"2C2F0F071B1F0E00"
130 PATTERN S#2,"00CCFE9E9CD878EF"
140 PATTERN S#3,"18F8F8F8060607C0"
150 Y1=Y+2:GOSUB 190
160 NEXT Y
170 M=M+2:IF M>3 THEN M=1
180 GOTO 20
190 SPRITE 0,(120,Y1),0,C
200 SPRITE 0,(120,Y1+1),0,C
210 RETURN
```

See Also: **PATTERN, MAG**

Statement	STOP

Function: Interrupt execution of a program for a while

Format: STOP

Description: Insert this statement into a program that behaves other than you expect. It will temporarily suspend execution of the program right where it was inserted.
By inserting this statement to various part of the program, you can keep an eye on the intermediate results step by step, and thus can find out what's wrong with the program.
For example, typing

PRINT "variable name" CR

will show you the intermediate value of the variable indicated by "variable name."
If you interrupt a program with this statement, the message

Break in "line number"

will appear on the screen.
The CONT statement will resume execution of the program right from immediately after the STOP statement if you didn't modify it.

Example :

```
LIST

10 FOR I=1 TO 9
20 FOR J=1 TO 9
30 PRINT I*J;
40 NEXT J:PRINT
50 STOP
60 NEXT I

RUN
 1  2  3  4  5  6  7  8  9
Break in 50
PRINT I,J
 1                    10
Ready
CONT
 2  4  6  8  10  12  14  16  18
Break in 50
```

Statement	VERIFYM

Function: Compares machine language programs saved on cassette with the program in memory

Format: VERIFYM "filename", verify start address

Description: This statement compares the machine language program on cassette indicated by "filename" and the program in memory. The message "Verify end" will be displayed if no difference has been detected between the two programs. If you specify the verify start address, the comparison will start from that address. If not, it will start from the address as previously indicated by CSAVEM. If "filename" is omitted, comparison will be done between the program in memory and the first machine language program found on the cassette.

Note: Filename must be the one you christened on save.

Example:

```
VERIFYM
*
Found
*
Ready
```

Statement	VPOKE

Function: Writes data to the VRAM (Video RAM)

Format: **VPOKE** address, data

Description: By writing data into the VRAM, you can draw characters or figures on the screen. The same with the text window.

Example :

```
10 FOR A=&H3C00 TO &H3FBF
20 VPOKE A,65
30 NEXT A
```

```
10 B=&H1800+32*8
20 FOR A=B TO B+7
30 VPOKE A,255
40 NEXT A
50 FOR C=0 TO 100:NEXT
60 FOR A=B TO B+7
70 VPOKE A,0
80 NEXT A
90 FOR C=0 TO 100:NEXT
100 GOTO 20
```

```
10 FOR V=15360 TO 15700
20 CURSOR 18,0:PRINT"VRAM ADDRESS"V
30 X=0:Y=10
40 VP=VPEEK(V)
50 VPOKE V+X+Y*40,VP
60 X=X+1:IF X=38 THEN X=0:Y=Y+1
70 NEXT V
```

Arithmetic Function ABS (absolute)

Function Gives the absolute value for the arithmetic expression X

Format ABS (X)

Description The absolute value of a value is the same with the value if the value is positive, and is equal to the negative of the value if the value is negative

Example

```
PRINT ABS(-5)
 5
Ready

PRINT ABS(3*(-6))
 18
Ready
```

Arithmetic Function ACS (arc-cosine)

Function Gives θ in COS (θ) inverse cosine function

Format ACS (X) X must be in the range –1 thru 1

Example

```
10 FOR S=-1 TO 1 STEP .5
20 X=ACS(S)
30 Y=DEG(X)
40 PRINT X,Y
50 NEXT S
RUN
 3.1415926536          180
 2.0943951024          120
 1.5707963268          90
 1.0471975512          60
 0                     0
```

Character String Function: ASC (ascii)

Function Converts characters into corresponding numbers (ascii codes)

Format ASC (character constant or character variable)
Only the first character of any string constant more than one character long will be converted

Description Computers don't understand characters and symbols as the way human beings do. They only understand numbers.
The way they understand characters and symbols, they have a set of numbers ranging from 32 thru 255 within them, each number corresponding to each character and symbol on your keyboard.
In this way they can distinguish the character A (which is 65 from the computer's point of view) from the character B (66).
Even though you supply a character string more than one character long, as in

? ASC ("BA")

this function only outputs the number corresponding to the first character of the string and ignores the rest.

Example Sort names in alphabetical order according to the first character of the names.
On RUN, displays corresponding ascii codes to the input characters and symbols.

PRINT ASC ("A") CR
65 ← the ascii code for 'A' is 65
PRINT ASC ("!") CR
33 ← the ascii code for '!' is 33

```
10 INPUT A$
20 O=ASC(A$)
30 PRINT O
40 GOTO 10
```

See also (CHR$)

SORT

```
10 INPUT "number of DATA";N
20 DIM A$(N)
30 FOR I=1 TO N:READ A$(I):NEXT I
40 D=N
50 D=INT(D/2)
60 IF D<1 THEN 180
70 DD=N-D
80 FOR I=1 TO DD
90 J=I
100 IF ASC(A$(J))<=ASC(A$(J+D)) THEN 160
110 N$=A$(J)
120 A$(J)=A$(J+D)
130 A$(J+D)=N$
140 J=J-D
150 IF J>=1 THEN 100
160 NEXT I
170 GOTO 50
180 FOR I=1 TO N:PRINT I,A$(I):NEXT I
200 DATA SUN,MERCURY,VENUS,EARS
210 DATA MOON,MARS,JUPITER,SATURN
220 DATA URANUS,NEPTUNE,PLUTO
230 DATA ASTEROID,MILKY WAY,GALAXY
RUN
number of DATA14
 1                    ASTEROID
 2                    EARS
 3                    GALAXY
 4                    JUPITER
 5                    MARS
 6                    MOON
 7                    MILKY WAY
 8                    MERCURY
 9                    NEPTUNE
 10                   PLUTO
 11                   SUN
 12                   SATURN
 13                   URANUS
 14                   VENUS
Ready
```

Arithmetic Function ASN (arc-sine)

Function: Gives θ in SIN (θ) — inverse sine function

Format: ASN (X) — X must be in the range −1 thru 1
The value of ASN (X) is in radian

Example:

```
10 FOR S=-1 TO 1 STEP .5
20 X=ASN(S)
30 Y=DEG(X)
40 PRINT X,Y
50 NEXT S
RUN
-1.5707963268        -90
-.5235987756         -30
 0                    0
 .5235987756          30
 1.5707963268         90
Ready
```

Arithmetic Function ATN (arc-tangent)

Function: Gives the inverse tangent — inverse tangent function

Format: ATN (X)

This function returns values within the range $-\frac{\pi}{2}$ thru $\frac{\pi}{2}$

Example:

```
10 X=ATN(1)
20 Y=DEG(X)
30 PRINT X,Y
RUN
 .7853981634          45
```

Character String Function **CHR$** **(character $)**

Function Converts ascii codes into corresponding characters or control codes

Format CHR$ (arithmetic expression to be converted)
Argument must be an integer in the range 32 thru 255
Floating point numbers are trancated to integers

Description Each character and symbol in your computer has associated with it a code number. And this function is used to convert a code number to its corresponding character or symbol.
Computers can perform numeric comparison, or can sort strings in alphabetical order because they have a fixed set of code numbers.
See the Character Code Table in the appendix. Characters and symbols are assigned code numbers greater than or equal to 32.
CHR$ function can be used for getting control codes too.

Usage PRINT CHR$ (65)

A ← the ascii code 65 corresponds to the character 'A'

Example Let's peak in the characters stored in your computer

```
10 FOR M=32 TO 255
20 PRINT CHR$(M);
30 NEXT M
```

```
RUN
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDE
FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
lmnopqrstuvwxyz{|}~
```

You can see the characters and symbols on your keyboard displayed on the screen. Those are the characters and symbols stored in your computer.
See the Character Code Table and verify that the code numbers on your screen and on the table are the same.

See also ASC

Arithmetic Function	COS	(cosine)

Function: A trigonometric function. gives the cosine of the arithmetic expression X

Format: COS (X) the argument X must be in radian

Usage: Let's find out the cosines of 0°, 30°, 60°, 90°.

Example:

```
10 FOR X=0 TO 90 STEP 30
20 A=COS(RAD(X))
30 PRINT X;TAB(10);A
40 NEXT X

RUN
 0          1
 30         .86602540379
 60         .50000000001
 90         0
Ready
```

Arithmetic Function	DEG	(degree)

Function: Gives the equivalent angle in degree of the arithmetic expression X in radian

Format: DEG (X)

Description: This function is the inverse of the RAD function and returns the equivalent angle in degree of the expression X (in radian) by multiplying it by 180/pi

Example:

```
PRINT DEG(0.26)

 14.896902673
Ready
```

Disk Function	DSKF	(disk free)

Function Gives the amount of free space on disk

Format DSKF

Description This function gives the amount of free space left on disk in units of K (kilo) bytes
The FILES statement can also be used for this purpose, but this function can be directly executed to see how many bytes are left on disk.

Example :

```
? DSKF
 65
Ready

? DSKF
 65
Ready
```

General Function	EOF	(end of file)

Function: Checks the end of sequential files

Format: **EOF** (# file descriptor)

Description. This function is used to check whether the file being read by the INPUT statement from disk reached its end. The function returns (-) if the end has been reached, and (0) if the file yet has data to be read.
The file indicated by the file descriptor must have been opened with the input mode.

Example:

```
10 OPEN F$ FOR INPUT AS #1
20 INPUT SN$
30 IF EOF(#1) THEN 70
40 INPUT #1,NA$,TL$
50 IF SN$<>NA$ THEN 40
60 PRINT NA$,TL$
70 CLOSE:END
```

Arithmetic Function EXP (exponent)

Function: Gives powers of e (the base for the natural logarithm)

Format: EXP (X)

Usage: Let's calculate e^1, e^2, and e^3 respectively.

Example:

```
10 FOR I=1 TO 3
20 X=EXP(I)
30 PRINT"EXP(";I;")=";X
40 NEXT I

RUN
EXP( 1)= 2.7182818284
EXP( 2)= 7.3890560987
EXP( 3)= 20.085536923
Ready
```

General Function FRE (free)

Function: Gives the amount of free memory.

Format: FRE

Description: This function gives the amount of free area among the area available to the BASIC.

Example:

```
PRINT FRE
 15000
Ready
```

Character String Function HEX$ (hexa $)

Function: Converts values of numeric expressions into equivalent hexadecimal numeric character strings

Format: **HEX$** (numeric variable or expression)

Description: The range of the argument to this function is from –32768 thru 32767, and the decimal fraction, if any, is truncated.
Computers handle numbers in hexadecimal as well as in decimal into the equivalent number in hexadecimal.
To convert a hexadecimal number into equivalent decimal number, type

PRINT &H "hexadecimal number"

To distinguish between decimal and hexadecimal numbers, add "/H' at the head of any hexadecimal numbers.
The hexadecimal number &H10 is equivalent to the decimal number 16.

Usage: Let's convert –10, –5, 0, 5, 10, 15 into equivalent hexadecimal numeric character strings.

Example:

```
10 FOR S=-10 TO 15 STEP 5
20 X$=HEX$(S)
30 PRINT S;"=";X$
40 NEXT S
RUN
-10=FFF6
-5=FFFB
 0=0
 5=5
 10=A
 15=F
Ready
```

Character String Function	INKEY$	(in key $)

Function: Gives a character entered from the keyboard

Format: INKEY$

Description: Gives the character corresponding to the key being pushed at the time of execution of this function. If no key is being pushed, it gives the null string.
This function cannot detect the RESET, BREAK and the FUNC keys.

* Null string is the character string zero-character long, and is represented as two consecutive double quotes (" ").

Usage:

```
10 X$=INKEY$
20 IF X$="" THEN 10
30 PRINT X$;
40 GOTO 10
```

The line 20 keeps watching whether a key is being pushed. If no key is being pushed, X$ is assigned the null string (the character string with nothing in it) and nothing is displayed on the screen.
The program thus keeps looping between the lines 10 and 20 (an infinite loop). If a key is pushed at this time, X$ is assigned the character corresponding to the key and the line 30 displays it. To get out of this infinite loop, type

```
25 IF X$="Z" THEN 100
100 PRINT "END":END
```

Hit the Z key to end this program.

Example:

Operation can be started by [GRAPH][GRAPH]

```
LIST

10 X=10:Y=11:CLS
20 CURSOR X,Y:PRINT" "A$" "
30 A$=INKEY$
40 IF A$="" THEN 30
50 IF A$=CHR$(29) THEN X=X+1
60 IF A$=CHR$(29) THEN X=X-1
70 IF X<0 THEN X=0
80 IF X>31 THEN X=31
90 GOTO 20
```

General Function	INP	(inport)

Function: Gives contents of the I/O area

Format: INP (address)

Description: This function gives data in the specified I/O port. In the following example, the data in the I/O port BE (hexa) is read into the variable A at line 10. The result may vary according to the current state of the computer.
I/O port numbers are predetermined by the system and must be in the range (&H00 thru &HFF).
This function is useful to detect, for example, the current state of the joystick which is an external I/O device.

Example:

```
10 FOR A=&HE0 TO &HE7
20 PRINT A;INP(A)
30 NEXT A

RUN
 224 128
 225 128
 226 208
 227 128
 228 0
 229 10
 230 195
 231 2
Ready
```

General Function	INPUT$	(input dollar)

Function: Reads a character string having the specified length from a sequential file indicated by the file descriptor.

Format: INPUT$

Example:

```
10 OPEN"TEL NOTE" FOR INPUT AS #1
20 A$=INPUT$(20,#1)
30 PRINT A$
40 CLOSE #1
```

Arithmetic Function	INT	(integer)

Function: Trancates the decimal fraction of given arguments to return the resulting integer.

Format: INT (x)

Description: Use this function to set the greatest integer not greater than a given value or the value of an arithmetic expression.
This function is useful for numeric truncation or rounding up.

Example:

```
10 FOR N=1 TO 2 STEP 0.1
20 L=INT(N+0.5)
30 PRINT N;"=";L
40 NEXT N

RUN
 1= 1
 1.1= 1
 1.2= 1
 1.3= 1
 1.4= 1
 1.5= 2
 1.6= 2
 1.7= 2
 1.8= 2
 1.9= 2
 2= 2
Ready
```

Character String Function **LEFT$** **(left $)**

Function: Gives a substring of the given character string having the given number of length counting from the left

Format: **LEFT$** (character string, length)

Description: Gives a substring of the given character string having the specified number of length counting from the left. A space counts as one character.

Usage: Let's store into M$ the substring consisting of the first 6th characters of the character string stored in A$, and display it onto the screen:

Example:

```
10 A$="coffee cocoa milk"
20 M$=LEFT$(A$,6)
30 PRINT M$

RUN
coffee
Ready
```

Up to 6th characters counting from the left

See also: RIGHT$

Character String Function LEN (length)

Function: Gives the number of characters in the given character string

Format: LEN (character string)

Description: This function gives the number of characters in the given character string. Special symbols and spaces count as one within a string enclosed by " ".

Usage: Let's count the number of characters in the character string contained in A$.

```
10 A$="SEGA PERSONAL COMPUTER"
20 PRINT LEN(A$)
RUN
 22
Ready
```

Note that a space counted as one.

Example:

```
LIST

10 A$="**********"
20 FOR I=1 TO LEN(A$)
30 PRINT LEFT$(A$,I)
40 NEXT I
RUN
*
**
***
****
*****
******
*******
********
*********
**********
Ready
```

Arithmetic Function	LGT	(log ten)

Function: Gives logarithm to the base 10 (common logarithm)

Format: LGT (X)

Usage: Let's find out the common logarithms of 10, 100 and 1000.

Example:

```
10 N=1
20 N=N*10
30 X=LGT(N)
40 PRINT"LGT(";N;")=";X
50 IF N<1000 THEN 20
RUN
LGT( 10)= 1
LGT( 100)= 2
LGT( 1000)= 3
Ready
```

Disk Function:	LOC (n)	(location counter)

Function: Gives the number of a record in files up to where the disk I/O has taken place so far

Format: LOC (#file descriptor)

Description: If the file indicated by the given file descriptor is a random file, this function returns the number of the record lately processed by the PUT or the GET statement.
If the file is a sequential file, the number of records processed from the file's open is returned.

Example: A = LOC (1)

```
30 X=LOC(1)
40 PRINT"DATA COUNT C";C
50 C=C+1
```

Disk Function:	LOF (n)	(length of file)

Function Gives the size of given files

Format LOF (#file descriptor)

Description This function gives the size of files in unit of sectors specified by the given file descriptor.
If the file is a random file, the returned value will be the largest record number in the file.
Files must be previously opened by the OPEN statement to be processed by this statement since the argument is a file descriptor.

Example : A = LOF (1)

```
10 REM
20 OPEN"TEL NO. " AS #1
30 M=LOF(1)
40 PRINT"DATA COUNT";M
50 INPUT "RECORD NO. ";R
60 IF R=0 THEN GOTO 110
70 GET#1,R:A$,0,20:B$,20,20
80 PRINT "NAME   ";A$
90 PRINT "TEL    ";B$
100 GOTO 50
110 CLOSE
```

Arithmetic Function	LOG	(log)

Function Gives logarithm to the base e (natural logarithm)

Format LOG (X)

Example :

```
10 FOR J=1 TO 3
20 X=LOG(J)
30 PRINT "LOG(";J;")=";X
40 NEXT J
RUN
LOG( 1)= 2.67468532E-11
LOG( 2)= .69314718057
LOG( 3)= 1.0986122886
Ready
```

Arithmetic Function	LTW	(log two)

Function Gives logarithm to the base 2

Format LTW (X)

Usage See LOG.

Example :

```
10 FOR J=2 TO 10 STEP 2
20 X=LTW(J)
30 PRINT"LTW(";J;")=";X
40 NEXT J

RUN
LTW( 2)= 1
LTW( 4)= 2
LTW( 6)= 2.5849625007
LTW( 8)= 3
LTW( 10)= 3.3219280949
Ready
```

Character String Function	MID$	(mid $)

Function Gives a substring of the given character string

Format **MID$** (character string, *m*, *n*)
This gives the substring of the character string starting from the *m*-th character and *n*-characters long

Description This function gives a substring of given number of length taking from the given character string. A space counts as one character

Usage Let's get the substring 5th-characters long starting from the 8th character of the character string

```
10 A$="coffee cocoa milk"
20 M$=MID$(A$,8,5)
30 PRINT M$
RUN
cocoa
Ready
```

length

Starting character

Length can be omitted as in MID$ (character string, *m*), in which case the result is the substring starting from the *m*-th character up to the end of the character string.

Example

```
PRINT MID$ "ABCDEFGHI",5)
EFGHI
Ready
```

See also LEFT$, RIGHT$

General Function	PEEK	(peek)

Function Gives content of memory at specified address

Format PEEK (address)

Description Use this function to peek at the content of memory at the desired address. The address must be in the range 0 thru 65535 (&H0 thru &HFFFF). The value is returned as one byte integer.

Example

```
10 FOR A=&H8000 TO &H8FFF STEP 8
20 PRINT RIGHT$("000"+HEX$(A),4);
30 FOR B=A TO A+7
40 C=PEEK(B)
50 PRINT " ";RIGHT$("0"+HEX$(C),2);
60 NEXT B:PRINT
70 NEXT A
```

Arithmetic Function	PI	(pi)

Function: Gives the ratio of circumference of circle to diameter

Format: **PI**

Description: 3.1415926536 is assigned to **PI** in your computer

Example:

```
PRINT PI
 3.1415926536
Ready


LIST

10 INPUT"radius ";A
20 S=A*A*PI
30 PRINT "area of circle ";S

RUN
radius 5
area of circle 78.53981634
Ready
```

Arithmetic Function	RAD	(radian)

Function: Gives the equivalent angle in radian of the arithmetic expression X in degree

Format: **RAD (X)**

Description: This function gives the equivalent angle in radian of the expression X (in degree) by multiplying it by pi/180

Usage: *Let's convert 0°, 15°, 30°, 45°, 60° into equivalent angles in radian:*

Example:

```
LIST

10 FOR I=0 TO 60 STEP 15
20 X=RAD(I)
30 PRINT"RAD(";I;")=";X
40 NEXT I

RUN
RAD( 0)= 0
RAD( 15)= .2617993878
RAD( 30)= .5235987756
RAD( 45)= .7853981634
RAD( 60)= 1.0471975512
Ready
```

Character String Function **RIGHT$** **(right $)**

Function Gives a substring of the given character string having the given number of length counting from the right

Format RIGHT$ (character string, length)

Description This function gives a substring of the given character string having the specified number of length counting from the right of the given string. A space counts as one character.

Usage Let's store into M$ the substring four-characters long (counting spaces) counting from the right of the character string stored in A$, and display it onto the screen.

Example

```
10 A$="coffee cocoa milk"
20 M$=RIGHT$(A$,4)
30 PRINT M$
RUN
milk
Ready
```

Up to 4th characters counting from the right

See also LEFT$ MID$

Arithmetic Function	RND	(random)

Function: Generates random numbers

Format: RND (x)

where

x > 0 : Generate random numbers successively
x = 0 : Initialize the random number series
x < 0 : Permute the random number series

Description: The random numbers senerated with this function are fractions greater than or equal to 0 and less than 1. Multiply them by appropriate factor to set random numbers having desired decimal places.

Example 1. Let's generate random numbers successively.

The random numbers have 11 decimal places.
To set a sequence of random numbers in the range 1 thru 500, multiply the returned values by 500 and add 1 to them.
If you need random numbers in whole number, use the INT statement to discard the decimal fraction.

```
10 FOR N=1 TO 5
20 R=RND(1)
30 PRINT R
40 NEXT N

RUN
 .74622984084
 .018117110489
 .51872376941
 .86514622469
 .7563778417
Ready
```

```
R = INT(RND(1) * 500 + 1)

10 FOR N=0 TO 5
20 R=INT(RND(1)*500+1)
30 PRINT R;:NEXT N

RUN
 173 450 334 145 6 227

Ready
```

Example 2. Let's generate random numbers in the range 0 thru 5 and add 1 to them.

```
10 FOR N=1 TO 6
20 R=INT(RND(-1)*6+1)
30 PRINT R;
40 NEXT N

RUN
 1 6 4 3 2 4

Ready
```

You see we have generated random numbers in the range 1 thru 6. You can use this as a fake dice.

See Also: INT

Arithmetic Function	SGN	(sign)

Function: Used to get the sign of numeric expressions/values

Format: SGN (X) returns –1 if X is negative
0 if X is 0
1 if X is positive

Example :

```
LIST

10 FOR I=-2 TO 2
20 N=SGN(I)
30 PRINT"SGN(";I;")=";N
40 NEXT I

RUN
SGN(-2)=-1
SGN(-1)=-1
SGN( 0)= 0
SGN( 1)= 1
SGN( 2)= 1
Ready
```

Arithmetic Function	SIN	(sine)

Function: A trigonometric function gives the sine of the arithmetic expression X

Format: SIN (X) the argument X must be in radian

Usage: Let's find out the sines of 0°, 30°, 60°, 90°

Example :

```
10 FOR TH=0 TO 90 STEP 30
20 S=SIN(RAD(TH))
30 PRINT TH;TAB(10);S
40 NEXT TH

RUN
 0          0
 30         .5
 60         .86602540379
 90         1
```

Character String Function	SPC	(space)

Function: Gives specified number of consecutive spaces

Format: SPC (length)

Description: This function must be used in one of the PRINT, LPRINT, or PRINT# statements.
The length must be an integer in the range 0 thru 55. In case length as a value has a decimal fraction, the fraction will be truncated entirely.

Usage:

```
10 PRINT"ABC";SPC(10);"XYZ"
RUN
ABC ←————————→ XYZ
Ready
        10 spaces
```

If some characters happen to be in the range specified to SPC, these characters will be erased from the screen.

See also: TAB

Arithmetic Function	SQR	(square root)

Function: Gives square roots of given valves

Format: SQR (X)

Usage: Let's calculate $\sqrt{2}$ and $\sqrt{3}$.

Example :

```
10 INPUT "FIGURE ";A
20 X=SQR(A)
30 PRINT"SQR.ROOT";A;"=";X
40 GOTO 10
RUN
FIGURE ? 2
SQR.ROOT 2= 1.41421356
FIGURE ? 3
SQR.ROOT 3= 1.73205081
FIGURE ?
```

General Function STICK

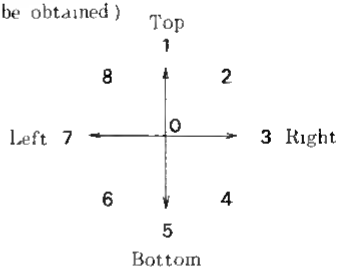
Function: Gives directions of joystick

Format: **STICK** (*f*)

Description:

Parameter (*f*): 1 = Joystick 1
2 = Joystick 2

(Value to be obtained)



Example:

```
10 REM JOY STICK TEST
20 B$="SHOOT":CLS
30 P1=STICK(1):P2=STICK(2)
40 S1=STRIG(1):S2=STRIG(2)
50 F1$="":F2$=""
60 IF P1=1 THEN F1$="UP     "
70 IF P1=3 THEN F1$="RIGHT  "
80 IF P1=5 THEN F1$="DOWN   "
90 IF P1=7 THEN F1$="LEFT   "
100 IF P2=1 THEN F2$="UP     "
110 IF P2=3 THEN F2$="RIGHT  "
120 IF P2=5 THEN F2$="DOWN   "
130 IF P2=7 THEN F2$="LEFT   "
140 IF S1>0 THEN F1$=F1$+B$+STR$(S1)
150 IF S2>0 THEN F2$=F2$+B$+STR$(S2)
160 CURSOR 10,10:PRINT CHR$(5)
170 CURSOR 1,10:PRINT"PLAYER 1 ";F1$
180 CURSOR 10,15:PRINT CHR$(5)
190 CURSOR 1,15:PRINT"PLAYER 2 ";F2$
200 GOTO 30
```

Character String Function STR$

Function: Converts numeric values to equivalent numeric character strings

Format: STR$ (numeric expression)

Description: Use this function to convert numeric values to equivalent numeric character strings. Note that the result cannot be used in numeric calculations

Usage: If the value is a positive number, a space is added at the head of the result.

```
LIST

10 A=1:B=3
20 D$=STR$(A)+STR$(B)
30 D=A+B
40 PRINT D$,D

RUN
 1 3                    4
```

Result of the line numbered 20 — Result of the line numbered 30

Note STR$ (A) turned the given number into the equivalent numeric character string. Adding characters doesn't do any calculation but putting them together.

Example:

```
LIST

10 A=1
20 A=A*10
30 IF A>1000 THEN END
40 D$=RIGHT$("   "+STR$(A),4)
50 PRINT D$
60 GOTO 20
RUN
   10
  100
 1000
Ready
```

See also: VAL

General Function	STRIG	(stick trigger)

Function: Gives states of the joystick trigger (the push button)

Format: STRIG (*f*)

Description:

			(Value to be obtained)
Parameter (*f*)	1 : Joystick 1	0 : off	
	2 : Joystick 2	1 : Trigger (left) ON	
		2 : Trigger (right) ON	
		3 : Trigger (left, right) ON	

Example:

```
10 REM JOY STICK TEST
20 B$="SHOOT":CLS
30 P1=STICK(1):P2=STICK(2)
40 S1=STRIG(1):S2=STRIG(2)
50 F1$="":F2$=""
60 IF P1=1 THEN F1$="UP     "
70 IF P1=3 THEN F1$="RIGHT  "
80 IF P1=5 THEN F1$="DOWN   "
90 IF P1=7 THEN F1$="LEFT   "
100 IF P2=1 THEN F2$="UP     "
110 IF P2=3 THEN F2$="RIGHT  "
120 IF P2=5 THEN F2$="DOWN   "
130 IF P2=7 THEN F2$="LEFT   "
140 IF S1>0 THEN F1$=F1$+B$+STR$(S1)
150 IF S2>0 THEN F2$=F2$+B$+STR$(S2)
160 CURSOR 10,10:PRINT CHR$(5)
170 CURSOR 1,10:PRINT"PLAYER 1 ";F1$
180 CURSOR 10,15:PRINT CHR$(5)
190 CURSOR 1,15:PRINT"PLAYER 2 ";F2$
200 GOTO 30
```

Character String Function	TAB	(tabulation)

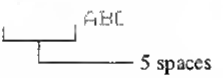
Function: Specifies a position from the left hand side of the screen

Format: TAB (number)

Description: This function must be used in one of the PRINT, LPRINT, or PRINT# statements
The number must be in the range 0 thru 37, and if it has a decimal fraction, the fraction will be truncated entirely
TAB (0) is equivalent to the extreme left hand side of the screen

Example:

```
10 PRINT TAB(5);"ABC"
RUN
     ABC
```



The TAB function will not erase characters which happen to be in the range specified to it
Remember this function since it is quite often used in the PRINT statement

See also: SPC

Arithmetic Function	TAN	(tangent)

Function: A trigonometric function gives the tangent of the arithmetic expression X

Format: TAN (X) the argument X must be in radian

Usage: Let's find out the tangents of values (in degree) input from the keyboard

Example:

```
10 INPUT "angle ";A
20 X=tan(A)
30 PRINT"TAN(";a;" )=";X
RUN
angle 30
TAN( 30 )= .57735026919
```

Character String Function TIME$

Function Sets and displays the inner clock

Format TIME$

Description Each computer has a clock in it.
And a very accurate, digital quartz clock is contained in your computer.
As soon as you switch on the power of your computer, the clock begins to tick with the interval of 1 second.
At the time of power-on, the clock is set to 00 00 00. Typing

PRINT TIME$ CR

will display the time elapsed from the time of power-on.

Example

For example, to set the time to 8 15 00, you type

TIME$ = "08 15 00" CR

the computer will answer

Ready

```
10 CLS
20 IF TIME$=T$ THEN 20
30 CURSOR 10,5:PRINT TIME$
40 T$=TIME$:BEEP
50 GOTO 20
```

Once you set the time, the clock keeps on going until you push the RESET button or turn off the power. Thus you can use your computer as a digital clock.

Character String Function **VAL** **(value)**

Function: Converts numeric character strings into equivalent numeric values

Format: VAL (numeric character string)

Description: The first character of the argument to this function must be either numeric o one of '+', '-', '$' or a space. Otherwise the result will be 0.
If the argument string contains a non-numeric character, then the rest of charac ters starting from that character will be ignored.
This is the inverse function of STR$.

Example :

```
10 A$="12345"
20 B$="11111"
30 C$=A$+B$                    ← Addition of numeric character strings
40 C=VAL(A$)+VAL(B$)           ← Addition of numeric values
50 PRINT C$
60 PRINT C
RUN
1234511111      ← Numeric character string
 23456          ← Numeric value
Ready
```

See also: STR$

General Function	VPEEK

Function: Gives the contents of the Video RAM

Format: VPEEK (Video RAM address)

Description: This function reads from the VRAM data for the characters or figures currently displayed on the screen. Refer to the address map for the URAM.

Example:

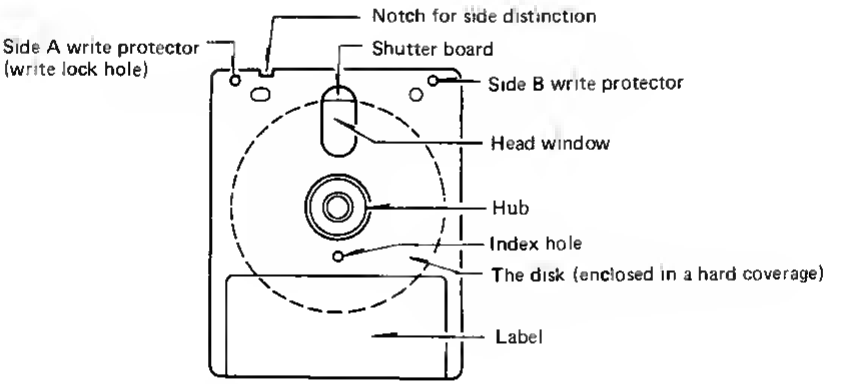
```
10 FOR V=15360 TO 15780
20 X=0:Y=10
30 VP=VPEEK(V)
40 VPOKE V+X+Y*40,VP
50 X=X+1:IF X=38 THEN X=0:Y=Y+1
60 NEXT V
```

CHAPTHR 6 FLOPPY DISK EXPLANATION

The media for the SF-7000 disk drive is a 3" floppy disk now called compact floppy disk designed for single head drive, though both sides A and B of the disk can be utilized

The compact floppy disk has a circular magneto-corted disk contained in a hard plastic coverage making it far handier than the traditional 5-1/4" floppy disks.

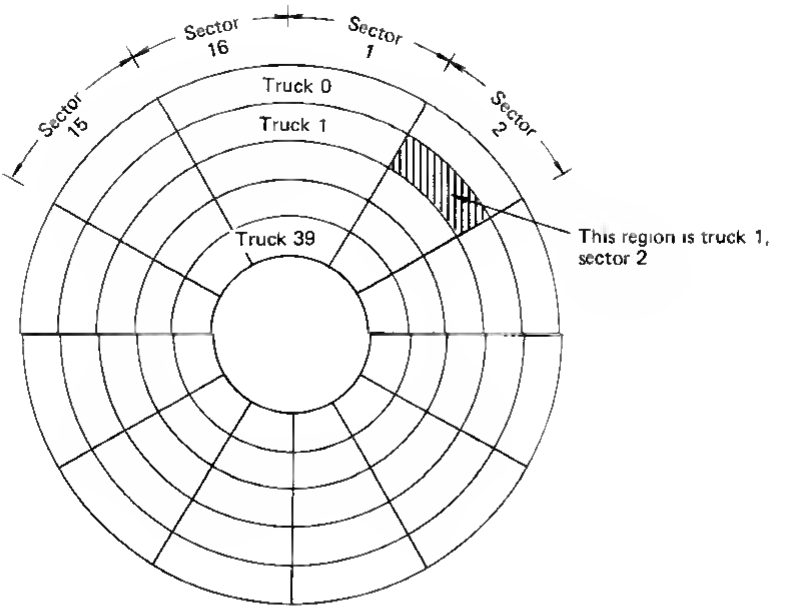
A compact floppy disk consists of a central hub for disk rotation, a head window and an index hole arranged around the hub for data transfer, write protect holes and a shutter board for data and disk board protection, and a few others.



Outline of a Compact Floppy Disk

On pushing this disk into the SF-7000 disk drive, the slider in the guide ditch on the side of the disk advances to open the protect shutter to greet the disk, and when the disk reaches to the bottom, the hub is automatically set to the drive, and so is head window to the head, protectors and the index hole to the detector

The rate of data transfer is faster than the traditional cassette tapes because the disk itself is rotated to place the head onto the desired position, making it a better device for faster data transfer



The surface of the media (a circular disk) is formatted to partition the available areas as shown in the figure. The surface of each disk is divided into 40 concentric circular strips called trucks numbered from 0 thru 39 counting from the outer-most truck.

The surface is further divided into 16 equi-angular sectors, and the cross section of a sector and a truck is simply called a "sector." Sectors are numbered from 1 thru 16 and each sector can contain 256 bytes of data. Note that a truck consists of 16 sectors.

Also a group of 4 sectors is called a cluster which is the unit utilised in file management.

Thus, since one surface of a disk holds 40 trucks each consisting of 16 256-byte sectors, the total amount of data recordable on one disk side is

256 bytes × 16 sectors × 40 trucks = 163,840 bytes

reaching nearly 16K bytes.

Since a space must be reserved on each side of one disk for file management purposes, the user available space is diminished by the amount of that space.

The available space is further diminished in disks such as a system disk that already contains some system programs (e.g., DISK BASIC).

1. The Disk Structure

Truck # (cylinder #)	Sector #	Contents
0	1	Disk information IPL in case of a system disk
	2 – 16	Reserved
1 – (*1)	1 – 16	System programs area User's area for non system disk
(*2) – 19	1 – 16	User's area used for storage of programs, etc
20	1 – 12	Directory (filename storage)
	13 – 16	FAT (sector 13 only)
21 – 39	1 – 16	User's area (program area) 77KB

(*1, *2 depend on the volume of system programs)

A system disk contains the filename IPL in the first sector on truck 0, which is loaded on a disk system boot into memory with address starting from FF00 up to FFFF

Sectors 1 thru 12 of truck 20 contains the directory and so does 13 thru 16 contain the FAT (File Allocation Table, a table containing location and order of files), each occupying 160 bytes In fact, only sector 13 is utilized for storage of the FAT

2. The Disk Capacity

Truck	Cluster	Sector	Byte
		1	256
	1	4	1,024
1	4	16	4,096
40	160	640	163,840

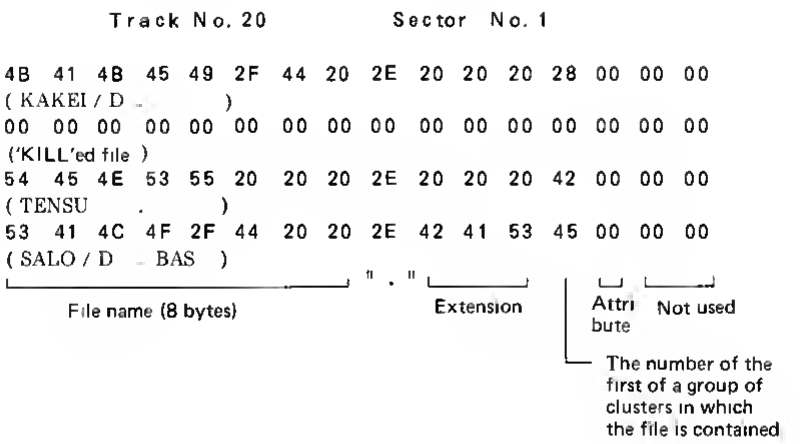
☆ Unit of data

☆ Unit of I/O

☆ Unit of file management

3. Directory

The directory is contained in sectors 1 thru 12 on truck 20.



File Dump Program

```
10 INPUT " Track  No.";T
20 INPUT " Sector No.";S
30 IF T<0 OR S<1 THEN BEEP2:GOTO 10
40 IF T>40 OR S>16 THEN BEEP2:GOTO 10
50 DSKI$ T,S;A$(0),0,128;A$(1),128,128
60 A$=A$(0):GOSUB 90
70 A$=A$(1):GOSUB 90
80 END
90 FOR I=1 TO 128 STEP 16
100 FOR J=I TO I+15
110 PRINT RIGHT$("0"+HEX$(ASC(MID$(A$,J,1))),2);" ";
120 NEXT J
130 PRINT
140 FOR J=I TO I+11
150 PRINT MID$(A$,J,1);
160 NEXT J
170 PRINT
180 IF INKEY$<>"" THEN 180
190 NEXT I
200 RETURN
```

Attributes

Value	Meaning	
00	Non ASCII format	[ ]
01	ASCII format	[ ]
02	Hexadecimal data	[*]
80	Non-ASCII read only	
81	ASCII read-only	
82	Hexadecimal read-only	

4. The FAT

Files are managed under units of clusters the total number of which is 160 (0 thru 159) on one side of a diskette. The FAT records the usage of each cluster on the side of the diskette. The FAT is 160-bytes long and is contained in sector 13 on truck 20, the remaining 96 bytes of the sector being unused.

1) The content of the FAT

≪A dump list≫

Track No 20 Sector No 13

```
FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE
FE FE FE FE FE FE FE FE 29 2A 28 2C 2D 2E 2F 30
31 32 33 34 C3 FF FF FF FF FF FF FF FF FF FF FF
FF FF 43 44 C2 46 C1 ....
```

≪Truck/cluster number list≫

NO.	TR.	CL.	FAT	NO.	TR.	CL.	FAT
0	0	00	FE	10	2	0A	FE
1	0	01	FE	11	2	0B	FE
2	0	02	FE	12	3	0C	FE
3	0	03	FE	13	3	0D	FE
4	1	04	FE	14	3	0E	FE
5	1	05	FE	15	3	0F	FE
6	1	06	FE	16	4	10	FE
7	1	07	FE	17	4	11	FE
8	2	08	FE	18	4	12	FE
9	2	09	FE	19	4	13	FE

NO.	TR.	CL.	FAT	NO.	TR.	CL.	FAT
20	5	14	FE	40	10	28	29
21	5	15	FE	41	10	29	2A
22	5	16	FE	42	10	2A	2B
23	5	17	FE	43	10	2B	2C
24	6	18	FE	44	11	2C	2D
25	6	19	FE	45	11	2D	2E
26	6	1A	FE	46	11	2E	2F
27	6	1B	FE	47	11	2F	30
28	7	1C	FE	48	12	30	31
29	7	1D	FE	49	12	31	32
NO.	TR.	CL.	FAT	NO.	TR.	CL.	FAT
30	7	1E	FE	50	12	32	33
31	7	1F	FE	51	12	33	34
32	8	20	FE	52	13	34	C3
33	8	21	FE	53	13	35	FF
34	8	22	FE	54	13	36	FF
35	8	23	FE	55	13	37	FF
36	9	24	FE	56	14	38	FF
37	9	25	FE	57	14	39	FF
38	9	26	FE	58	14	3A	FF
39	9	27	FE	59	14	3B	FF

Meaning of each byte datum in the FAT

FAT data	Meaning
0 – 9FH	Clusters in use, having succeeding clusters the value is the number of the cluster succeeding to it
C1H – C4H	The last clusters in use, the lower 4-byte indicates the number of sectors the cluster is using.
FEH	Reserved, can't be used for files (IPL, system programs, directory, etc.)
FFH	Unused clusters

For example, you can see from the dump list of the directory that the first cluster of a group of clusters allocated to the file "KAKEI/D" is 28H. The 28Hth data in the FAT table is 29H, meaning the following cluster is 29H. The 29Hth data in the FAT table is 2AH, and so on.

5. System Disks

At system disk startup, the loader first loads the content of the first sector on track 0 into memory in the address range FF00 thru FFFF, then checks the first 4 bytes starting from the address FF00. It then displays onto the screen the Filename immediately following the first 4 bytes, and jumps to the IPL program starting from the address FF20.

Thus if you want to create a system disk, the first thing you have to do is to save the system disk id code, the filename, and the IPL program onto the first sector on track 0 of the disk.

Address	Data		Contents
FF00	53	S	System disk identification code
1	59	Y	
2	53	S	
3	3A	:	
FF04	20		File name
	64	d	
	69	i	
	73	s	
	6B	k	
	20		
	42	B	
	41	A	
	53	S	
	49	I	
	43	C	
	00		
	:		
FF1F			

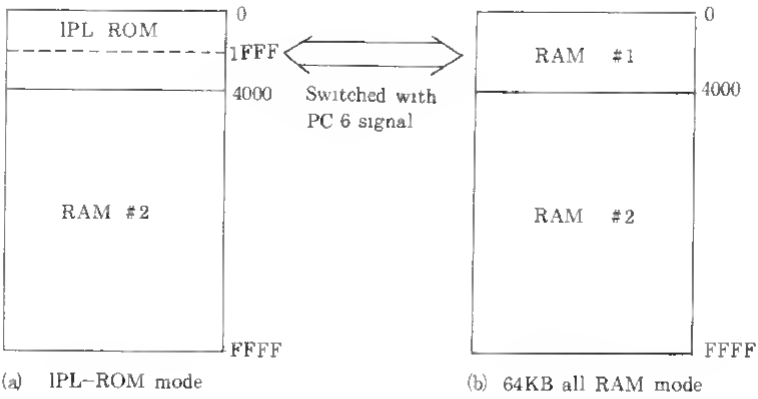
6 IPL PROGRAM

```
;
;          IPL  PROGRAM
;
                                    (Referencial example)
           ORG    OFF20H
           CALL   8          ; track 0
           JP     C , 0      ; error then boot
           LD     DE , 0     ; load start address
           LD     BC , 0101H ; start sector = 1     track = 1
           LD     HL , 1009H ; end   sector = 10H   track = 9
LOOP:
           CALL   10H
           JP     C , 0
           PUSH   HL
           OR     A
           SBC    HL , BC
           POP    HL
           JR     Z , EXIT
           INC    D
           INC    B
           LD     A , B
           CP     11H
           JR     NZ , LOOP
           LD     B , 1
           INC    C
           JR     LOOP
EXIT:
           LD     A , 3      ; Motor off
           OUT    (0E7H) , A
           LD     A , DH     ; select
           OUT    (0E7H) , A ; 0~3FFF → RAM#1
           JP     0
```

SF-7000 HARDWARE

Memory Map

SF-7000 has equipped with it a 64KB RAM and a 8KB ROM for the IPL (Initial Program Loader).

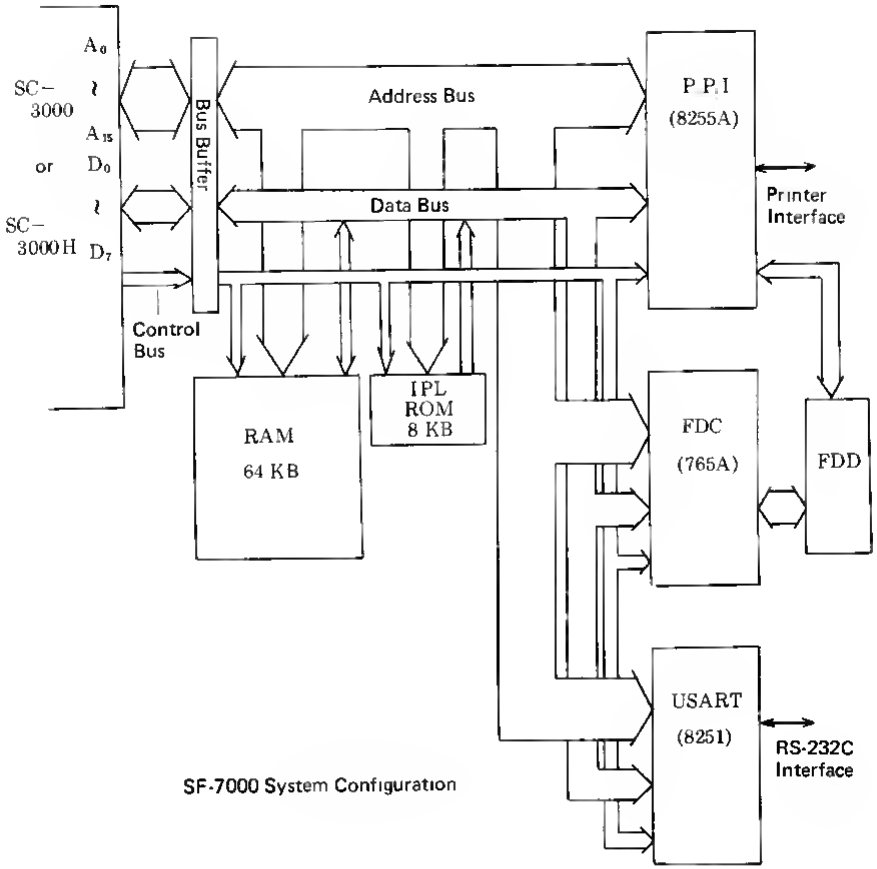


SF-7000 memory map

The memory map for the SF-7000, either on power-on or during execution of system utility programs is as shown in Fig (a) where the IPL is ready for its execution.

At the SF-7000 startup, the system programs (e.g., Disk BASIC) are loaded onto RAM starting from the address 0, then after the load has completed, the address 0 thru 3FFF is switched to RAM and the execution is passed to the starting address of a user program.

The interchange between ROM and RAM in the address range 0 thru 3FFF is handled by the P.P.I., PC6 signal.



SF-7000 System Configuration

The SC-3000 Series and the Controller for the SF-7000

Port Map (SC-3000)

P.P.I (8255)

Port address		Mode	Assignment
&H DC	(220)	PA Input	Keyboard (input)
DD	(221)	PB Input	Keyboard (input)/printer, cassette
DE	(222)	PC Output	Keyboard scan/printer, cassette
DF	(223)	Control register	

VDP (TMS 9918A)

Port address		Assignment
&H BE	(190)	VDP data register
BF	(191)	VDP command register

PSG (SN 76489)

Port address		Assignment
&H 7F	(127)	PSG

Port Map (SF-7000)

PPI (8255A)

Port address		Mode	Assignment
&H E4	(228)	PA input	FDC/printer check
E5	(229)	PB output	Printer data output (parallel)
E6	(230)	PC output	FDC/printer control
E7	(231)	Control register	

FDC (765AC)

Port address		Assignment
&H E0	(224)	Status register
E1	(225)	Data register

USART (8251)

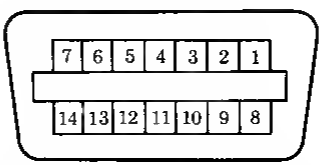
Port address		Assignment
&H E8	(232)	USARTD (data)
E9	(233)	USARTC (command)

The Printer Interface

The SF-7000 has equipped with it a printer interface for 8-bit parallel data.

Pin number	Signal	Function		Signal direction
1	$\overline{STB}$	Strobe signal		→ Printer
2	D_1	Data 1	Bit 0	→
3	D_2	2	1	→
4	D_3	3	2	→
5	D_4	4	3	→
6	D_5	5	4	→
7	D_6	6	5	→
8	D_7	7	6	→
9	D_8	8	7	→
10	N C			
11	BUSY	Printer busy signal		←
12	N.C.			
13	N.C.			
14	GND	Voltage 0V		

Amphenol 14-pin female type



Connector socket outlook and its pin numbers
(rear view of the SF-7000)

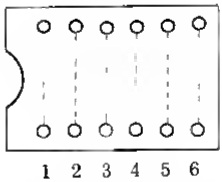
The RS-232C Serial Interface

The SF-7000 has equipped with it an RS-232C interface which enables you to communicate with other computers or send data for output to an RS-232C printer. At shipment the baud rate is set to 300 baud, but you can change it selectively from 300 up to 9600 by replacing the jumper lead on the printed-circuit board inside the SF-7000.

Caution: Never set more than one jumpers.

Dip-switch settings for baud rates

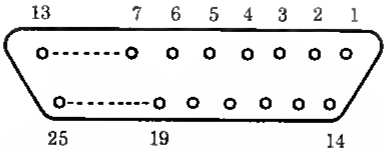
1 only	Baud rate	9,600 baud
2 only		4,800 baud
3 only		2,400 baud
4 only		1,200 baud
5 only		600 baud
6 only		300 baud



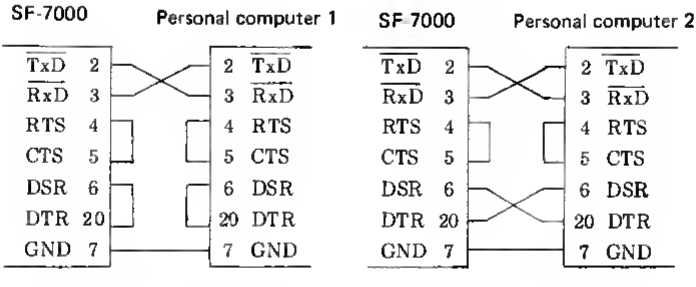
The serial interface 8251A (USART) is used for the RS-232C interface of the SF-7000 that, under control of the CPU, enables such format settings as synchronous/asynchronous communications mode switching, number of bits par one character (character length), parity check enabling, and so on.

The RS-232C Interface

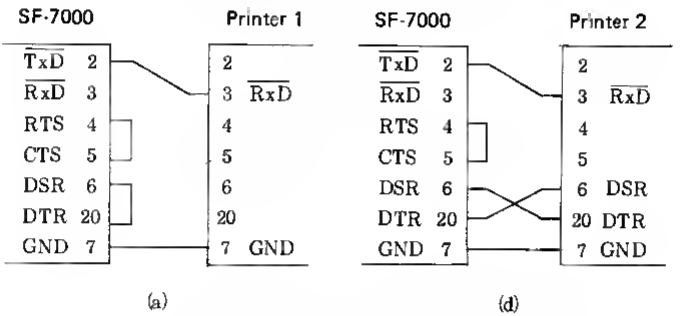
Terminal number	Signal name	Function
1	FG (Frame GND)	Chassis ground
2	$\overline{\text{TxD}}$ ($\overline{\text{Transmitted Data}}$)	Serial data send
3	$\overline{\text{RxD}}$ ($\overline{\text{Received Data}}$)	Serial data receive
4	RTS (Request To Send)	Carrier control
5	CTS (Clear To Send)	Data send control
6	DSR (Data Set Ready)	Modem Ready signal in
7	SG (Signal GND)	Ground
20	DTR (Data Terminal Ready)	Connects to modem CD



Outlook of the RS-232C connector socket and its pin numbers
(Rear view of the SF-7000)



Two examples connecting the SF-7000 with other personal computers
(Bi-directional communication)



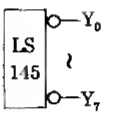
Two examples connecting the SF-7000 with printers
(Uni-directional communication)

In Fig. (a), RTS and CTS are shortened as well as DSR and DTR, yielding the signals to be sent unchecked. Thus, the data transfer must be controlled according to the receiver's capacity.

P.P.I. Functional Table

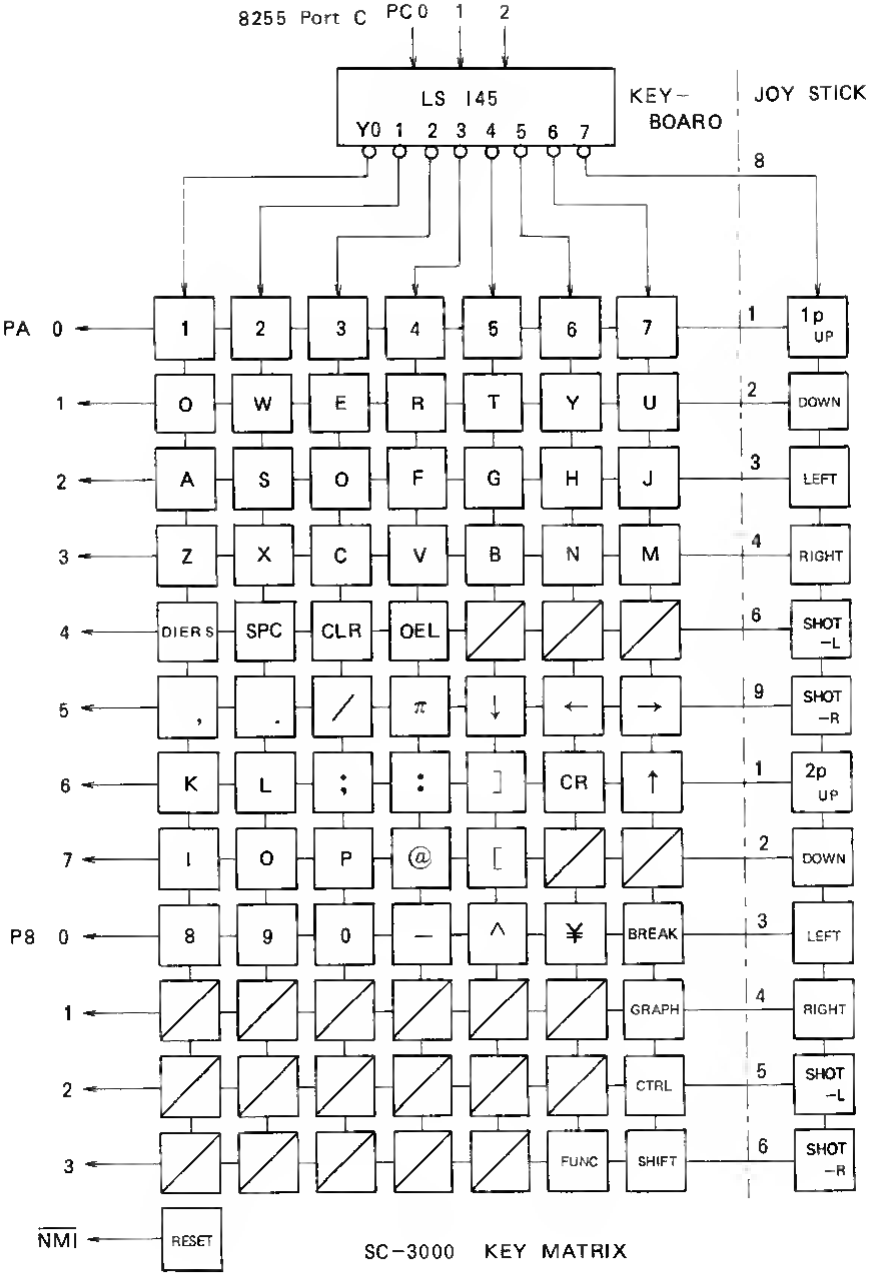
P.P.I (SC-3000)

Terminal	Assignment
PA0	Keyboard input
1	
2	
3	
4	
5	
6	
7	
PB0	Keyboard input
1	
2	
3	
4	$\overline{\text{CONT}}$ (to the external terminal B-11)
5	FAULT } The serial printer
6	BUSY } SP-400 input
7	Cassette tapes Input
PC0	Keyboard raster (output)
1	
2	
3	N, C
4	Cassette tapes Output
5	Data
6	$\overline{\text{RESET}}$ } The serial printer
7	$\overline{\text{FEED}}$ } SP-400 input



P.P.I. (SF-7000)

Terminal	Signal name	Function
PA0	FDCINT	INT signal input from FDC
1	BUSY	BUSY from Centronics printer
3	–	———
4	–	———
5	–	———
6	–	———
7	–	———
PB0	Data 1	Data outputs to Centronics printer
1	Data 2	
2	Data 3	
3	Data 4	
4	Data 5	
5	Data 6	
6	Data 7	
7	Data 8	
PC0	$\overline{\text{INUSE}}$	INUSE signal to FDD
1	$\overline{\text{MOTOR}}$	MOTOR ON signal to FOD
2	TC	TC signal to FDD
3	RESET	Reset signal to FDC
4	–	———
5	–	———
6	$\overline{\text{RON SEL}}$	Switching between IPL ROM and RAM
7	$\overline{\text{STROBE}}$	$\overline{\text{STROBE}}$ to Centronics printer

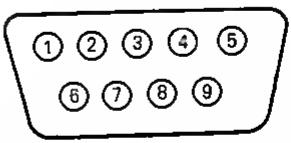

8255 Port C PC0 1 2
LS 145
Y0 1 2 3 4 5 6 7
KEY-BOARO
JOY STICK
8
PA 0
1 2 3 4 5 6 7
1 1p UP
1 O W E R T Y U 2 DOWN
2 A S O F G H J 3 LEFT
3 Z X C V B N M 4 RIGHT
4 DIER S SPC CLR OEL 6 SHOT -L
5 , . / π ↓ ← → 9 SHOT -R
6 K L ; :] CR ↑ 1 2p UP
7 I O P @ [2 DOWN
P8 0 8 9 0 — ^ ¥ BREAK 3 LEFT
1 GRAPH 4 RIGHT
2 CTRL 5 SHOT -L
3 FUNC SHIFT 6 SHOT -R
NMI RESET
SC-3000 KEY MATRIX

1. The Joystick Terminal

Two connector sockets, JOY1 and JOY2, for joysticks are supplied at the rear end of the SF-3000, each connected to the P P.I inside the SF-3000.

Pin number	Function
1	Lever — UP
2	Lever — DOWN
3	Lever — LEFT
4	Lever — RIGHT
5	———
6	Trigger LEFT
7	———
8	Common
9	Trigger RIGHT

AMP 9-pin male type

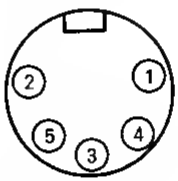


Outlook of the joystick connector socket and its pin numbers

2. Video/audio Signal Output Terminal

Pin number	Function
1	Audio signal
2	Video signal
3	GND
4	GND
5	GND

DIN 5-pin female type

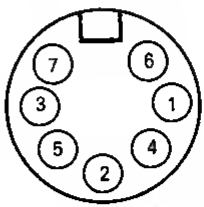


Outlook of the VIDEO connector socket and its pin numbers

3. Serial Printer Connector Tip

Pin number	Function
1	$\overline{\text{FAULT}}$
2	BUSY
3	DATA
4	$\overline{\text{RESET}}$
5	$\overline{\text{FEED}}$
6	GND
7	NC

DIN 7-pin female type



Outlook of the serial printer connector socket and its pin numbers

SC-3000 Extention Slot Connector Terminal

Side A (lower)		Side B (upper)	
Pin number	Signal name	Pin number	Signal name
1	A_0	1	+5V
2	A_1	2	+5V
3	A_2	3	$\overline{\text{DSRAM}}$
4	A_3	4	$\overline{\text{CEROM2}}$
5	A_4	5	$\overline{\text{MEMR}}$
6	A_5	6	$\overline{\text{MEMW}}$
7	A_6	7	$\overline{\text{I/OR}}$
8	A_7	8	$\overline{\text{I/OW}}$
9	A_8	9	N.C
10	A_9	10	$\overline{\text{MREQ}}$
11	A_{10}	11	CON
12	A_{11}	12	$\overline{\text{RAS1}}$
13	A_{12}	13	$\overline{\text{CAS1}}$
14	A_{13}	14	RAM A7
15	D_0	15	$\overline{\text{RAS2}}$
16	D_1	16	$\overline{\text{CAS2}}$
17	D_2	17	$\overline{\text{MUX}}$
18	D_3	18	A_{14}
19	D_4	19	A_{15}
20	D_5	20	N.C
21	D_6	21	GND
22	D_7	22	GND

$A_0 - A_{15}$ CPU address bus

$D_0 - D_7$ CPU data bus

$\overline{\text{DSRAM}}$ OPEN if S-RAM for SC-3000 is used, +5V if not

$\overline{\text{CEROM2}}$ Select memory 0 – 7FFF

$\overline{\text{MEMR}}$ CPU signal $\overline{\text{MREQ} \cdot \text{RD}}$

$\overline{\text{MEMW}}$ CPU signal $\overline{\text{MREQ} \cdot \text{WR}}$

$\overline{\text{I/OR}}$ CPU signal $\overline{\text{IORQ} \cdot \text{RD}}$

$\overline{\text{I/OW}}$ CPU signal $\overline{\text{IORQ} \cdot \text{WR}}$

$\overline{\text{MREQ}}$ CPU signal

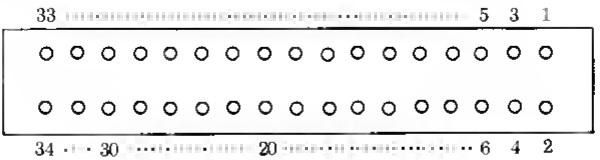
CON P.P I PB4

$\overline{\text{RAS 1}}$, $\overline{\text{CAS 1}}$ (8000 ~ BFFFH) D–RAM Control signal

$\overline{\text{RAS 2}}$, $\overline{\text{CAS 1}}$ (C000 ~ FFFFH) D–RAM Control signal

$\overline{\text{MUX}}$ D–RAM Control signal

Connector socket to the computer



Pin number	Signal name	Pin number	Signal name
1	A_0	2	A_1
3	A_2	4	A_3
5	A_4	6	A_5
7	A_6	8	A_7
9	A_8	10	A_9
11	A_{10}	12	A_{11}
13	A_{12}	14	A_{13}
15	A_{14}	16	A_{15}
17	D_0	18	D_1
19	D_2	20	D_3
21	D_4	22	D_5
23	D_6	24	D_7
25	GND	26	$\overline{\text{I/OR}}$
27	GND	28	$\overline{\text{I/OW}}$
29	GND	30	$\overline{\text{MEMR}}$
31	GND	32	$\overline{\text{MEMW}}$
33	GND	34	$\overline{\text{MREO}}$

CHAPTER 7 SUPPLEMENT

Variables and Arrays

Numeric variables	A, B, ~ Z, AA, AB, ~ ZZ
Numeric array	A 0, A 1 ~ A 9 (Subscript, ~) Up to 3 – DIM A (15), B (5, 5), AC (3, 3, 3)
String variables	A$, A B$, A 1$
String array	(Subscript, ~) Up to 3 – DIM A$ (15), B$ (5, 5), AC$ (3, 3, 3)

- For variable names, the first character is an alphabetic character and then after, alphabetic characters or numerics. Although any number of characters is acceptable, separation is made by the beginning 2 characters.

- The names of variables and arrays may be the same.

(Range of numeric variables and arrays)

±9.9999999999E−99
≀
±9.9999999999E+99

(Range of string variables and array)

Character length 0～31

CONSTANT

Numeric constant

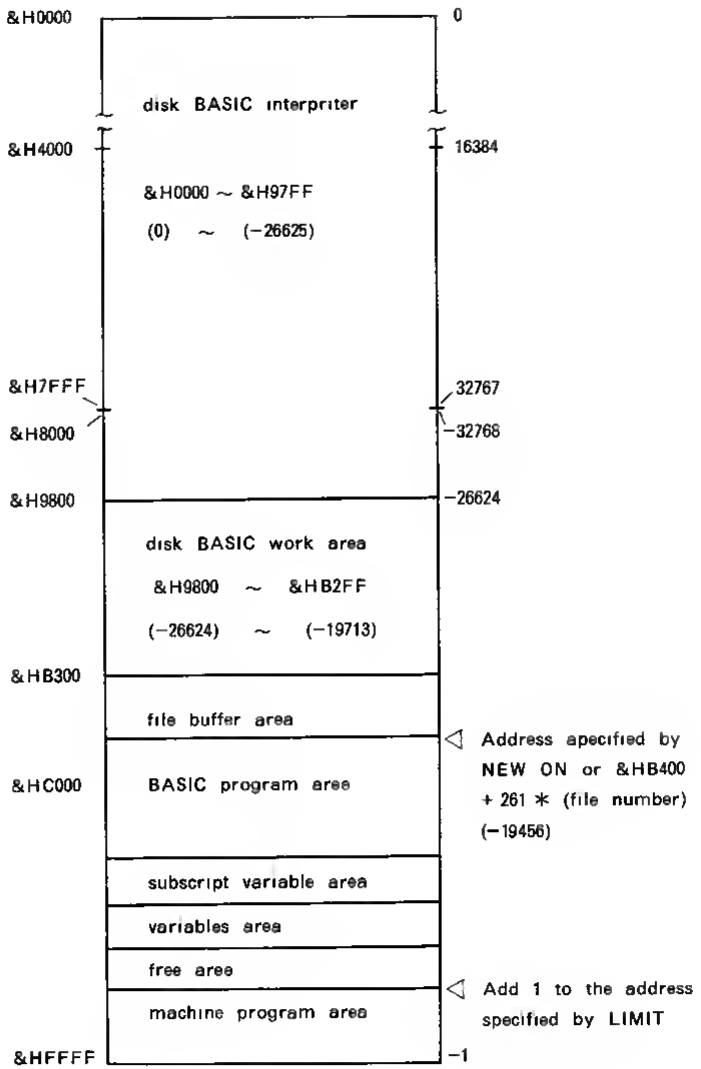
Integer form	Example 3, −2, 99926768
Decimal form	Example 0.2, .3, −5.3, 86.0
Exponent form	Example 3, E99, −6E3, 0.3E+5, 4E−82
Hexadecimal form	&H Hexadecimal value ~ 0000～FFFF Example &H64 same as 100 &HFFFF same as −1

String constant

Use double quotation to indicate " enclosed by "

Example "ABC" → Character ABC
"" → Character NULL
"""" → Character "
"A3""64" → Character A3"64

Main Memory Map



Relationship between File Buffer Area and BASIC Program Area

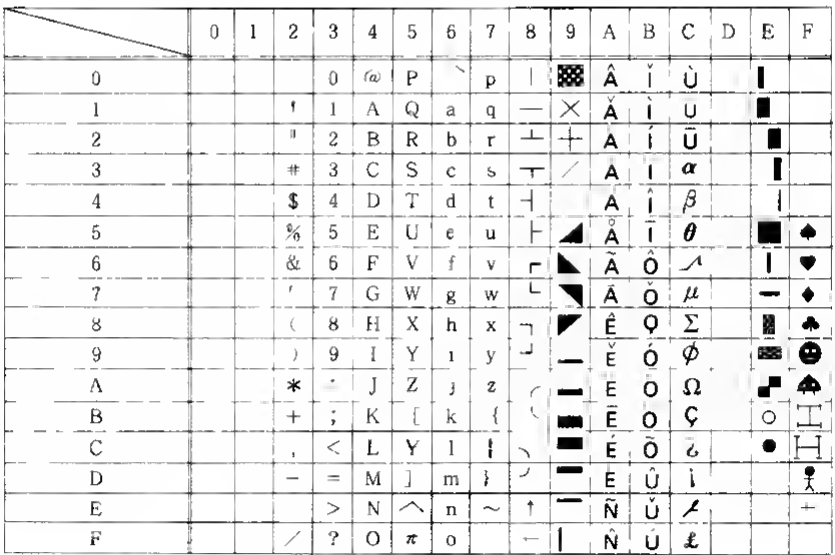
Number of File	BASIC Program Start Address	Free Area
0	B400H	19452
1	B505H	19191
2	B60AH	18930
3	B70FH	18669
4	B814H	18408
5	B919H	18147
6	BA1EH	17886
7	BB23H	17625
8	BC28H	17364

At the time DISK BASIC starts, the file number, specified is 3.

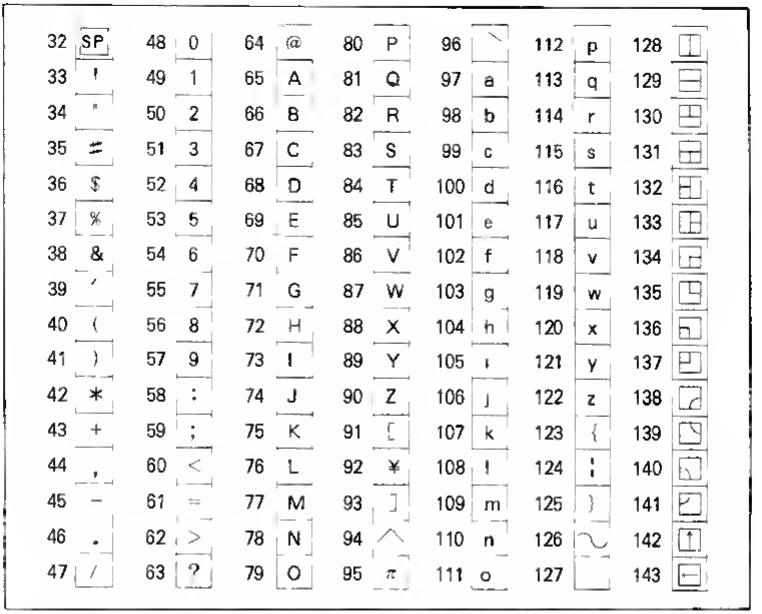
LIMITATIONS

Contents	Limitation
Characters taken into the inside from the screen	256 characters
Character numbers usable for actual text image by reserved words converted from line buffer	256 characters
Character numbers which can be handled as character string	255 characters
Level number such as operator priority, etc.	32 levels
Area for string operation	300 characters
FOR ~ NEXT nesting level number	16 levels
GOSUB, RETURN nesting level number	8 levels

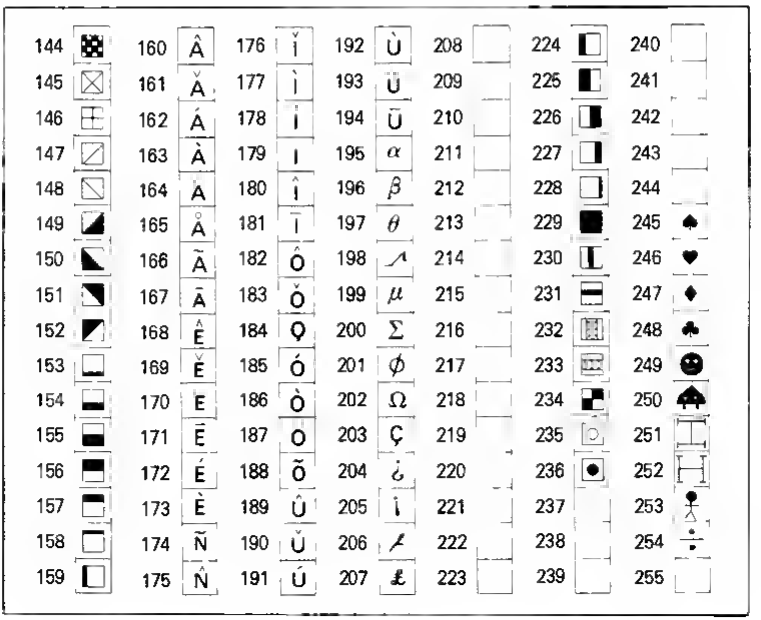
CHARACTER SET



CHARACTER CODE



32	SP	48	0	64	@	80	P	96	`	112	p	128			
33	!	49	1	65	A	81	Q	97	a	113	q	129			
34	"	50	2	66	B	82	R	98	b	114	r	130			
35	#	51	3	67	C	83	S	99	c	115	s	131			
36	$	52	4	68	D	84	T	100	d	116	t	132			
37	%	53	5	69	E	85	U	101	e	117	u	133			
38	&	54	6	70	F	86	V	102	f	118	v	134			
39	'	55	7	71	G	87	W	103	g	119	w	135			
40	(	56	8	72	H	88	X	104	h	120	x	136			
41	)	57	9	73	I	89	Y	105	i	121	y	137			
42	*	58	:	74	J	90	Z	106	j	122	z	138			
43	+	59	;	75	K	91	[	107	k	123	{	139			
44	,	60	<	76	L	92	¥	108	l	124	¦	140			
45	−	61	=	77	M	93	]	109	m	125	}	141			
46	.	62	>	78	N	94	^	110	n	126	~	142			
47	/	63	?	79	O	95	π	111	o	127		143			



144		160	Â	176	Ĭ	192	Ù	208		224		240			
145		161	Ă	177	Ì	193	Ü	209		225		241			
146		162	Á	178	Ī	194	Ū	210		226		242			
147		163	À	179	I	195	α	211		227		243			
148		164	Ā	180	Î	196	β	212		228		244			
149		165	Å	181	Ī	197	θ	213		229		245	♠		
150		166	Ã	182	Ô	198	ʌ	214		230		246	♥		
151		167	Ā	183	Ŏ	199	μ	215		231		247	♦		
152		168	Ê	184	Ǫ	200	Σ	216		232		248	♣		
153		169	Ĕ	185	Ó	201	ϕ	217		233		249			
154		170	Ē	186	Ò	202	Ω	218		234		250			
155		171	Ẽ	187	Ō	203	Ç	219		235		251			
156		172	É	188	Õ	204	¿	220		236		252			
157		173	È	189	Û	205	ì	221		237		253			
158		174	Ñ	190	Ŭ	206	≠	222		238		254			
159		175	Ñ	191	Ú	207	£	223		239		255			

Error Message

Message	Meaning
Array name	Argument to DIM statement not array
Bad file mode	Attempted to process file against its mode
Bad file number	Wrong file descriptor
Command parameter	Wrong argument to command
Can't continue	Can't continue with CONT statement
Device not ready	Printer not connected or out of order
Disk full	No free space in disk, can't SAVE
Disk I/O	Attempted to write on write protected disk (protect hole open), invalid I/O
Disk offline	Attempted to access disk when offline (not inserted into drive)
Division by zero	Denominator was zero
Extra ignored	Wrong input data to INPUT statement, extra data ignored
File already exists	File already exists under the name supplied as argument to NAME (rename)
File already open	Attempted to open a file which is already open
File not found	There is no file as indicated
File not opened	Attempted to access file that is not open
File write protected	Write attempt to read-only file
FOR nesting	More than 8 levels of depth in FOR–NEXT
FOR variable name	Non numeric variable to FOR (character type or array)
Function buffer full	Attempted to define more than 8 user functions
Function parameter	Wrong argument to a function
GOSUB nesting	Nested subroutines with more than 16 levels
Illegal direct	Cannot execute direct statement
Illegal line number	Line number is improper
Input past end	Attempted to read sequential file past end of file
Line image too long	Too many characters in one line (RENUM, etc.)
Line number over	Line numbers exceeded 65535 in AUTO or RENUM
Number of subscripts	Number of subscripts is unusual

Message	Meaning
NEXT without FOR	No corresponding FOR to the NEXT statement
N-formula too complex	Too complex numeric expression
No program	Attempted to SAVE while no program in text buffer
Out of data	READ tried to read from a DATA statement having no data in it
Out of memory	
Overflow	Numeric overflow in arithmetic expression or in value
PRT: 2 not selected	Attempted to take hard copy (HCOPY) of graphics window without switching to appropriate printer (I/O unit)
Redim'd array	Attempted to re-declare an array
Redo from start	Wrong data to INPUT statement need re-input
RETURN without BOSUB	RETURN statement executed having no corresponding GOSUB
S-formula too complex	String expression too complex
Stack overflow	Too many parentheses, figure too complex for PAINT, or a user function is recursively defined
Statement parameter	Wrong argument to statement
String too long	Character string was more than 255-characters long
Syntax	Syntax error
System	The BASIC interpreter made an error (not possible)
Type mismatch	Type mismatch between data and variable (value, string)
Undef'd array	Attempted to ERASE undefined array
Undef'd function	Attempted to call undefined user function
Undef'd line number	No line under line number (RENUM, GOTO, GOSUB, IF-THEN, RESTORE, RUN)
Unprintable	Unknown error
Value of subscript	Subscript wrong or out of range
Verifying	Error during comparison between program in memory and program on cassette

SC-3000 PARALLEL PRINTER ESCAPE SEQUENCE MODIFICATION PROCEDURE

* This manual explains the configuration and modification procedure of the parallel printer escape sequence table

1. Escape Sequence Table Configuration

Disk BASIC for SC-3000 and SF-7000 does not limit parallel printers (RP80, etc.) It references the escape sequences used in the interpreter from the table. Thus, various types of printers can be used only by modifying the table.

1.1 Escape sequence functions

Disk BASIC provides nine escape sequences.

1.1.1 Right margin setting

This escape sequence sets the right printing margin. Usually, this may be the same as the maximum column number of the printer. In most printers, this is automatically set in the initial state.

1.1.2 Horizontal tabulation setting

This escape sequence sets the tabulation interval used to separate the printed characters by a comma (,) specified in a LPRINT statement. The interpreter sets this interval to 20 columns and the same value must be set on the printer. If another value is set on the printer, the TAB(n) result for the associated line will not be guaranteed. For a printer not having the horizontal tab setting function, tabulation by a comma (,) is not performed or correct tabulation cannot be expected.

1.1.3 Standard character setting

This escape sequence sets the standard characters printed. Usually, specify pica characters. This setting is not necessary for a printer having no character selection function.

1.1.4 Bidirectional printing

This escape sequence specifies bidirectional printing to reduce print head travelling time for ordinary character printing. This setting is not necessary for a printer having no bidirectional printing function.

1.1.5 1/6-inch line space setting

This escape sequence specifies the line pitch for ordinary character printing. An arbitrary pitch can be set; it may not be 1/6 inch.

1.1.6 Unidirectional printing

Since graphic printing may have misalignment on both ends of a line, it is better to return the print head to its home position after printing each line. This escape sequence specifies unidirectional printing.

1.1.7 1/9-inch line space setting

This escape sequence specifies the line pitch for graphic printing. This must be equal to the vertical character length (for EPSON's RP80, this is 1/9 inch (8 dots)).

1.1.8 Bit image (256 bytes) setting

This escape sequence specifies printing the subsequent 256 bytes as graphic data.

1.1.9 Bit image (512 bytes) setting

This escape sequence specifies printing the subsequent 512 bytes as graphic data.

1.2 Escape sequence table structure

Each of the nine sequences explained in Section 1.1 is in a frame consisting of a 1-byte sequence length and 15-byte image. This sequence table resides in the disk BASIC boot loader program, and is transferred to the interpreter working area when the system is started. Therefore, this table can be applied to various types of printers by modifying the boot loader program area. The following table lits the addresses and sample data for RP80 and RX80.

	Address	Escape sequence	Hexadecimal data
1	70H-7FH	ESC 'Q' + B0	03, 1B, 51, 50
2	80H-8FH	ESC 'e' 0 + 20	04, 1B, 65, 00, 14
3	90H-9FH	ESC 'P'	02, 1B, 50
4	A0H-AFH	ESC 'U' + 0	03, 1B, 55, 00
5	B0H-BFH	ESC '2'	02, 1B, 32
6	C0H-CFH	ESC 'U' + 1	03, 1B, 55, 01
7	D0H-DFH	ESC '3' + 24	03, 1B, 33, 1B
8	E0H-EFH	ESC 'K' + 0 + 1	04, 1B, 4B, 00, 01
9	F0H-FFH	ESC 'K' + 0 + 2	04, 1B, 4B, 00, 02

- In the hexadecimal data, the first byte indicates the sequence length and the subsequent bytes indicate the sequence image. If the sequence image does not occupy 15 bytes, the remaining bytes are filled with 00H.
- The indicated address is the relative address in the 256 bytes of the boot loader program (track 0, sector 1). Data in the area whose address is less than 70H must not be modified.

2. Escape Sequence Table Modification Procedure

The escape sequence table can be modified using a DSKOS statement or the special tool, in any case, it may cause the system disk to be destroyed. Therefore, be sure to generate a backup disk before modification.

2.1 Table modification with DSKOS statement

After picking up all data to be modified, link these data to arbitrary character string variables. When writing data in a specific area in a sector, DSKOS clears all data except that in this area, therefore, it is necessary to read the boot loader program contents beforehand with other variables, then add the value of the character variable for the generated data to the contents.

2.2 Table modification with special tool

Execute ESC-TBL.BAS attached to disk BASIC Version 1.0p and input the necessary data as required.

When ESC-TBL.BAS is executed, the sequence table in the boot loader program is read and displayed. Enter a digit 1-9 in reply to "ESC number" and enter "/" when modification is completed. "*" is displayed at the beginning of the associated escape sequence; to modify this data, enter thenew data in reply to "ESC data ". The entered data must be hexadecimal digit strings, delimited with a space ( ). (In this case, the sequence length need not be checked and the sequence image can be directly used.) The data entered by CR is checked for whether it contains non-hexadecimal data and whether its length exceeds 15 bytes. If the entered data is illegal, reentry is requested. If the data is legal, it is displayed in the line having *. Check this data for correctness. If it is correct, enter "Y" (or "y") in reply to "data OK? (Y/N)", if not, enter "N" (or "n"). When "N" is replied, the previous data is redisplayed and "ESC data " is prompted again. When "Y" is replied, * disappears from the beginning of the line and "ESC number " is prompted again.

After modification is completed, enter "/" in reply to "ESC number ". When "/" is replied, "really (Y/N)" is displayed to request for entry. When "N" is replied, "ESC number " is prompted again. When "Y" is replied, all table elements containing the modified data are written in the boot program and this operation is terminated. Since the sequence modification is valid from the next system start, a BOOT command must be executed to continue processing.

Example

Modification to FP100 (PC8801 mode)

Modified items 1) Right margin setting ESC 'Q' + 136 ($1B_H$, 51_H, 88_H)

2) Horizontal tab setting(*) ESC 'l' +20 + 40 + 60 + 80 + 100 + 120 + 135 + NUL
($1BH$, $14H$, $28H$, $3CH$, $50H$, $64H$, $78H$, $87H$, $00H$)

1 Start disk BASIC, then execute ESC-TAB.BAS.

2 Modified item 1)

Enter "1 CR " in reply to "ESC number:".

3 Enter "1B△51△88 CR " in reply to "ESC data:".

4 In reply to "data OK (Y/N)", enter "Y CR " if the displayed data is correct, if not, enter "N CR " to return to step 3 above.

5 Modified item 2)

Enter "2 CR " in reply to "ESC number:".

6 Enter "1B△14△28△ ... △00 CR " in reply to "ESC data:".

7 Same as step 4 above. If the data is incorrect, return to step 6 above.

8 Enter "/" in reply to "ESC number:".

9 Enter "Y" in reply to "really (Y/N)".

10 End of modification

* Though FP100 defines the horizontal tab ending value as 135, column 136 cannot be set actually. Therefore, for the 6th tab by a comma (,) in a LPRINT statement for a line, the data is printed from the end of the line. So, do not program tabulation over more than one line.

SAMPLE PROGRAM

VRAM address of test

```
10 REM ---- VRAM address of te t ---
20 FOR V=15360 TO 15700
30 CURSOR30,0:PRINT V
40 X=0:Y=10
50 VP=VPEEK(V)
60 'POKE V+X+Y*40,VP
70 X=X+1:IF X=38 THEN X=0:Y=Y+1
80 NEXT V
```

VRAM address, graphcs SCREEN

```
10 REM -VRAM address, graphcs SCREEN -
20 SCREEN 2,2:CLS
30 PRINT"H"
40 SCREEN 1,1:CLS
50 AD=&H0000+8
60 FOR A=AD TO AD+7
70 DA=VPEEK(A)
80 PRINT HEX$(DA)
90 NEXT A
```

SPRITE HIT

```
10 REM --- SPRITE HIT ---
20 SCREEN 2,2 :CLS
30 PATTERNS#0,"389CDAFFDE9C0838"
40 PATTERNS#4,"0202020226262A2418"
50 X=80:Y=90
60 LINE(163,0)-(163,90),10
70 MAG2
80 SPRITE 0,(X,Y),0,5
90 SPRITE 2,(150,88),4,8
100 X=X+1
110 S=INP(&HBF) AND &H20
120 IF S<>0 THEN GOTO 140
130 GOTO 80
140 CLS
150 CIRCLE(120,90),50,6,,.3
160 CURSOR110,85:PRIN"HIT !"
170 FOR W=0 TO 100:NEXT W
180 CLS:GOTO 20
```

Bar graph

```
10 REM       - Bar graph   --
20 CLS
30 FOR I=0 TO 5
40 PRINT CHR$(65+I);"-";
50 INPUT "number of 0 to 20 ";A(I)
60 IF A(I) 00RA(I) 20 THEN 40
70 NEXT I
80 CLS:N=20:FOR Y=0 TO 20
90 IF YMOD5=1 THEN CURSOR0,Y:PRINT N;:
N=N-5
100 CURSOR",Y:PRINT CHR$(128);
110 NEXT Y
120 FOR X=3 TO 36
130 CURSORX,21:PRINT CHR$(158);
140 NEXT X
150 FOR X=6 TO 35 STEP 3
160 CURSORX,22:PRINTCHR$(65+(X-6)/7);
170 FOR Y=20 TO 21-A((X-6)/3) STEP -1
180 IF A((X-6)/3)=0 THEN 200
190 CURSORX,Y:PRINT"*";
200 NEXTY
210 NEXTX
220 GOTO 220
```

Line graph

```
10 REM  - --  Line graph ------
20 CLS
30 FOR I=0 TO 7
40 PRINT CHR$(65+I);" ";
50 INPUT "number of 0 to 100 ";A(I)
60 IF A(I) 00RA(I) 100 THEN 40
70 NEXT I
80 INPUT "Bar graph 1  or Line graph<2
>";B
90 IF B 10RB 2 THEN BEEP2:GOTO 80
100 GOSUB 260
110 ON B GOTO 120,170
120 FOR X=41 TO 230 STEP 24
130 CURSORX,23:PRINTCHR$(65+(X-41)/24)
140 COLOR,4,(X-1,40)-(X+6,40+A((X-41)/
24))
150 NEXT X
160 GOTO 160
170 CURSOR41,23:PRINT"A"
180 CIRCLE(40,40+A(0)),2,4
190 PSET(40,40+A(0)),4
200 FOR X=64 TO 230 STEP 24
210 COLOR1:CURSORX,23:PRINTCHR$(65+(X-
64)/24+1)
220 CIRCLE(X,40+A((X-64)/24+I)),2,4
230 LINE -(X,40+A((X-64)/24+1))
240 NEXT X
250 GOTO 250
260 SCREEN 2,2:COLOR1,15:CLS
270 POSITION(0,191),0,1
280 LINE(31,170)-(31,40),1
290 LINE -(240,40)
300 CURSOR12,144:PRINT"100"
310 CURSOR18,94:PRINT"50"
320 CURSOR24,44:PRINT"0"
330 RETURN
```

COLOR AND SPRITE

```
10 REM ---- COLOR  AND SPRITE -----
20 SCREEN 2,2:COLOR,I,(0,0)-(255,191),
1:CLS
30 PATTERN S#0,"FFFFFFFFFFFFFFFF"
40 PATTERN S#1,"FFFFFFFFFFFFFFFF"
50 PATTERN S#2,"FFFFFFFFFFFFFFFF"
60 PATTERN S#3,"FFFFFFFFFFFFFFFF"
70 PATTERN S#4,"A5A5A5A5A5A5A5A5"
80 PATTERN S#5,"A5A5A5A5A5A5A5A5"
90 PATTERN S#6,"A5A5A5A5A5A5A5A5"
100 PATTERN S#7,"A5A5A5A5A5A5A5A5"
110 FOR M=0 TO 2
120 RESTORE 190:MAG M:CLS
130 FOR I=0 TO 5.652 STEP .628
140 READ N,C
150 X=SIN(I)*80+127
160 Y=COS(I)*65+90
170 SPRITE N,(X,Y),0,C
180 NEXT I
190 DATA 0,2,3I,8,1,4,30,I7,2,10
200 DATA 29,3,3,7,28,6,4,5,27,11
210 FOR I=0 TO 6.28 STEP .1
220 X=SIN(I)*80+127
230 Y=COS(I)*65+90
240 SPRITE 9,(X,Y),4,15
250 NEXT I,M
260 GOTO 110
```

```
10 REM ---- color clock ----
20 SCREEN 2,2:COLOR,15,(0,0)-(255,191)
,15:CLS:MAG1
30 POSITION(0,191),0,1
40 PATTERN S#0,"071F3F7F7FFFFFFF"
50 PATTERN S#1,"FFFFFF7F7F3F1F07"
60 PATTERN S#2,"E0F8FCFEFEFFFFFF"
70 PATTERN S#3,"FFFFFFFEFEFCF8E0"
80 PATTERN S#4,"A5A5A5A5A5A5A5A5"
90 PATTERN S#5,"A5A5A5A5A5A5A5A5"
100 PATTERN S#6,"A5A5A5A5A5A5A5A5"
110 PATTERN S#7,"A5A5A5A5A5A5A5A5"
120 DEFFNS=SIN(RAD(VAL(RIGHT$(TIME$,2)
)*6))*80+127
130 DEFFNC=COS(RAD(VAL(RIGHT$(TIME$,2)
)*6))*65+90
140 DEFFNS1=SIN(RAD(VAL(MID$(TIME$,4,2
))*6))*53.3+127
150 DEFFNC1=COS(RAD(VAL(MID$(TIME$,4,2
))*6))*43.3+90
160 DEFFNS2=SIN(RAD(VAL(LEFT$(TIME$,2)
)*30+VAL(MID$(TIME$,4,2))/2))*40+127
170 DEFFNC2=COS(RAD(VAL(LEFT$(TIME$,2)
)*30+VAL(MID$(TIME$,4,2))/2))*32.5+90
180 CIRCLE(134,182),8,1,1,,,BF
190 LINE(126,167)-(142,182),,BF
200 FOR I=1 TO 12
210 X=SIN(RAD(30*I))*80+135
220 Y=COS(RAD(30*I))*65+82
230 PSET(X,Y-10),I
240 LINE-(X-9,Y+5)
250 LINE-(X+9,Y+5)
260 LINE-(X,Y-10)
270 PSET(X-9,Y-5)
280 LINE-(X,Y+10)
290 LINE-(X+9,Y-5)
300 LINE-(X-9,Y-5)
310 LINE(135,82)-(X,Y),1
320 NEXT I
330 BLINE(107,90)-(163,75),,BF
340 CURSOR111,86:PRINTTIME$
350 SPRITE 5,(FNS,FNC),0,13
360 BEEP:WW=VAL(RIGHT$(TIME$,2))
370 SPRITE 6,(FNS1,FNC1),0,8
380 SPRITE 7,(FNS2,FNC2),0,7
390 GOTO 330
```

```
10 REM ----------
20 REM  clock
30 REM ------ ----------
40 CLS:PRINT"              00:00:00"
50 INPUT "what time ";T$
60 IF T$="" THEN 80
70 TIME$=T$
80 SCREEN 2,2:CLS:PRINTCHR$(16);
90 POSITION (0,0),0,0
100 COLOR ,15,(0,0) (255,191)
110 RT=115
120 LINE (30,0)-(225,191),2,BF
130 POSITION (128,96),0,1
140 BCIRCLE(0,0),74,15,RT,,,BF
150 FOR I=1 TO 12
160 TH=RAD((3-I)*30)
170 Y=67*SIN(TH)*RT+4
180 X=67*COS(TH)-7
190 IF I =10 THEN X=X-3
200 COLOR 2
210 CURSOR X 6,Y :PRINTI;
220 NEXT
230 L1=43 : L2=RT*L1
240 L3=55 : L4=RT*L3
250 L5=55 : L6=RT*L5
260 H=VAL(LEFT$(TIME$,2))
270 M=VAL(MID$(TIME$,4,2))
280 HR=RAD(30*H+M/2)
290 HX=LI*SIN(HR)
300 HY=L2*COS(HR)
310 H1=H
320 MR=RAD( 6*M)
330 MX=L3*SIN(MR)
340 MY=L4*COS(MR)
350 M1=M
360 REM
370 REM   ------ ------------
380 REM
390 S=VAL(RIGHT$(TIME$,2))
400 IF S=S1 THEN 390
410 T$=TIME$
420 H=VAL(LEFT$(T$,2))
430 M=VAL(MID$(T$,4,2))
440 S1=S
450 SR=RAD( 6*S)
460 SX=L5*SIN(SR)
470 SY=L6*COS(SR)
480 IF M=M1 THEN 580
490 MR=RAD( 6*M)
500 MX=L3*SIN(MR)
510 MY=L4*COS(MR)
520 M1=M
530 HR=RAD(30*H + M/2)
540 HX=LI*SIN(HR)
550 HY=L2*COS(HR)
560 BLINE(0,0)-(XM,YM),2
570 BLINE(0,0)-(XH,YH),2
580 BLINE(0,0)-(XS,YS),2
590 LINE(0,0)-(HX,HY),2
600 LINE(0,0)-(MX,MY),2
610 LINE(0,0)-(SX,SY),2
620 BLINE( 24,-87)-(23,-95),15,BF
630 CURSOR -24,-88:COLOR I
640 PRINT T$;
650 GOSUB 720
660 XH=HX : XM=MX : XS=SX
670 YH=HY : YM=MY : YS=SY
680 GO TO 780
690 REM
700 REM ----------
710 REM
720 IF M<59 THEN 790
730 IF S<57 THEN 790
740 SOUND 1,440,15
750 FOR J=0 TO 10
760 NEXT
770 SOUND 0
780 MF=-1
790 IF MF=-1 THEN 840
800 IF(V<8)THEN BEEF 1 : BEEP 0
810 IF V<0 THEN 840
820 SOUND 1,880,V
830 V=V-2
840 IF M=1 THEN 880
850 IF S>1 THEN 880
860 SOUND 1,880,15
870 V=14 : MF=0
880 RETURN
890 REM----------
900 F=220:SOUND 1,,15
910 FOR I=1 TO 4
920 C1=17*RND(1)+1:Y=RND(1)*40
930 D0=0:D=2*I
940 FOR X=-31 TO 245 STEP 4*I
950 D0=D0+D:Y=Y+D0
960 IF ABS(D0)>7*I THEN D=-D
970 SPRITE I,(X,Y),I*4+12,C1
980 SOUND 1,RND(1)*F+F
990 NEXT:F=F*2
1000 NEXT
1010 BEEP0:RETURN
```

```
10 '==        Block break game
20  DEFINT A-Z
30  CLEAR 10,1
40 CLS:I=5:SC=0
50 CURSOR 0,0:PRINT"** block break game***
   HIGH SCORE: ";:PRINT HS
60 CURSOR21,1:PRINT"SCORE:";:GOSUB880
70 FOR I=2 TO 21:CURSOR 0,I:PRINT"X"
80 CURSOR 25,I:PRINT"X":NEXT I
90 CURSOR 1,2:FOR I=1 TO 24:PRINT"X";:NEXT I
100 FOR Y=4 TO 10 STEP 2
110 CURSOR 2,Y:FOR I=1 TO 11:PRINT"■■ ";:NEXT I,Y
120 A=16:CURSOR A,22:PRINT"~~~"
130 XX=INT(RND(1)*13)+10:YY=11:D=INT(RND(1)*2)+2
140 BEEP1:FOR I=1 TO 150:NEXT I:BEEP0
150 FOR I=1 TO 300:NEXT I
160 A$=INKEY$
170 IF A$=CHR$(28) THEN 210
180 IF A$  CHR$(29) THEN 230
190 IF A 1 THEN 270
200 A=A-2:CURSOR A,22:PRINT"~~  ":GOTO 230
210 IF A>31 THEN 230
220 A=A+2:CURSOR A-2,22:PRINT"  ~~"
230 ON D GOTO 240,290,350,410
240 PP=VPEEK((YY-1)*40+XX+1+2+&H3C00):GOSUB 830
250 ON P GOTO 260,500,620
260 GOSUB 800
270 XX=XX+1:YY=YY-1:GOSUB 810
280 GOTO 160
290 PP=VPEEK((YY+1)*40+XX+1+2+&H3C00):GOSUB 830
300 ON P GOTO 310,520,640
310 GOSUB 800
320 XX=XX+1:YY=YY+1:GOSUB 810
330 IF YY=22 THEN 460
340 GOTO 160
350 PP=VPEEK((YY+1)*40+XX-1+2+&H3C00):GOSUB 830
360 ON P GOTO 370,540,660
370 GOSUB 800
380 XX=XX-1:YY=YY+1:GOSUB 810
390 IF YY=22 THEN 460
400 GOTO 160
410 PP=VPEEK((YY-1)*40+XX-1+2+&H3C00):GOSUB 830
420 ON P GOTO 430,560,680
430 GOSUB 800
440 XX=XX-1:YY=YY-1:GOSUB 810
450 GOTO 160
```

```
460 GOSUB 800:BEEP 2
470 FOR I=1 TO 300:NEXT I
480 T=T-1:IF T=0 THEN 710
490 GOTO 130
500 X=XX:Y=YY-1:GOSUB 780
510 X=XX+1:D=2:GOTO 580
520 X=XX:Y=YY+1:GOSUB 780
530 X=XX+1:D=1:GOTO 580
540 X=XX-2:Y=YY+1:GOSUB 780
550 X=XX-1:D=4:GOTO 580
560 X=XX-2:Y=YY-1:GOSUB 780
570 X=XX-1:D=3
580 GOSUB 810
590 SC=SC+1:GOSUB 820
600 IF SC=44 THEN 710
610 GOTO 160
620 IF XX=34 THEN D=4:GOTO 700
630 D=2:GOTO 700
640 IF XX=34 THEN D=3:GOTO 700
650 D=1:GOTO 700
660 IF XX=1 THEN D=2:GOTO 700
670 D=4:GOTO 700
680 IF XX=1 THEN D=1:GOTO 700
690 D=3
700 BEEP:GOTO 160
710 FOR I=1 TO 200:BEEP 1:BEEP 0:NEXT I
720 CURSOR 1,22:PRINTSPC(30)
730 CURSOR 5,22:INPUT " once more ? [Y/N]? ";B$
740 IF B$="N" THEN END
750 IF B$<>"Y" THEN BEEP 2:GOTO 720
760 IF SC>HS THEN HS=SC
770 GOTO 40
780 CURSOR X,Y:PRINT SPC(3):GOSUB 800
790 BEEP:RETURN
800 CURSOR XX,YY:PRINT" ":RETURN
810 CURSOR XX,YY:PRINT"●":RETURN
820 CURSOR 27,1:PRINT SC:RETURN
830 P$=CHR$(PP)
840 IF P$=" " THEN P=1:GOTO 870
850 IF P$="■" THEN P=2:GOTO 870
860 P=3
870 RETURN
880 CURSOR 27,1:PRINT SC:RETURN
```

Commands, Statements and Functions

Commands

No	Command name	Function	Page
1	AUTO	Automatically generates line numbers	57
2	BOOT	Re-loads system programs from disk	58
3	CLOAD	Loads programs from cassette	59
4	COMLOAD	Receives programs via RS-232C interface	60
5	COMSAVE	Sends programs via RS-232C interface	60
6	CONT	Resumes interrupted programs	61
7	CSAVE	Saves programs onto cassette	62
8	DELETE	Deletes parts of programs	63
9	FILES	Displays names of files on disk	64
10	LFILES	Outputs to printer names of files on disk	65
11	LIST	Displays programs onto the screen	66
12	LLIST	Outputs programs to printer	67
13	LOAD	Loads programs from disk	68
14	MAXFILE	Declares number of simultaneously openable files	69
15	MERGE	Merges programs on disk with program in memory	70
16	NEW	Deletes/resets programs and variables in memory	71
17	NEWON	Sets starting address for program area	71
18	RENUM	Re-sequence line numbers of programs	72
19	RUN	Begin execution of programs	73
20	SAVE	Saves programs onto disk	74
21	UTILITY	Enters disk utility program	75
22	VERIFY	Compares program in memory with that saved on cassette	76

No.	Statement	Function	Page
1	BCIRCLE	Deletes by circle	77
2	BEEP	Beeps	78
3	BLINE	Deletes by line	79
4	CALL	Calls machine language subroutines	80
5	CIRCLE	Draws circles	81
6	CLOADM	Loads machine language programs from cassette	83
7	CLOSE	Closes specified files	84
8	CLS	Clears screen	85
9	COLOR	Specifies color	86
10	COMSET	Used for RS-232C interface specifies data length, sets/resets parity bit	88
11	CONSOLE	Sets scroll limit for the screen, click sound, upper/lower case distinction, and select printer	89
12	CSAVEM	Saves machine language programs onto cassette	90
13	CURSOR	Sets cursor position on the screen	91
14	DATA	Specifies data to be read by the READ statement	92
15	DEF FN	Defines user functions	93
16	DIM	Declares arrays	94
17	DSKI$	Does direct read from disk	95
18	DSKO$	Does direct write to disk	97
19	END	Terminates program execution	99
20	ERASE	Clears declared arrays	100
21	FOR TO-NEXT STEP	Repeat lines between FOR and NEXT STEP means increment to FOR	101
22	GET	Read into variable data in random files	102
23	GOSUB—RETURN	Jumps to line specified by line number Returns from subroutine	104
24	GOTO	Jumps to line specified by line number	105
25	HCOPY	Outputs to printer current screen image	106
26	IF—THEN	Conditionally jumps to/executes statements	107

No	Statement	Function	Page
27	INPUT	Input value or character string from keyboard	108
28	INPUT #	Input value or character string from indicated sequential file or from RS-232C interface	109
29	KILL	Erase files from disk	111
30	LET	Assigns value to variable (omittable)	112
31	LIMIT	Sets end address for BASIC program area	113
32	LINE	Draws line	114
33	LOADM	Loads machine language programs into specified memory area	115
34	LPRINT or L?	Output to printer value or character string Characters and symbols must be enclosed in ' " '	116
35	MAG	Set sprite width	117
36	NAME	Change name of programs on disk	120
37	ON—GOSUB — RETURN	Conditionally jumps to subroutine indicated by line number Returns from subroutine	121
38	ON—GOTO	Conditionally jumps to line indicated by line number	122
39	OPEN	Opens disk files	123
40	OUT	Outputs data to output board	124
41	PAINT	Paints specified region	125
42	PATTERN	Sets character and sprite pattern	126
43	POKE	Write data in memory	130
44	POSITION	Sets coordinate for graphics screen	131
45	PRESET	Deletes by points	132
46	PRINT or ?	Displays value or string onto screen Characters and symbols must be enclosed in ' " '	133
47	PRINT #	Write value or character string to specified sequential file or RS-232C interface	134
48	PSET	Draws point	136
49	PUT	Write value of expression to random file	137
50	READ	Reads data specified in DATA statement	139
51	REM	Marks comment	140
52	RESTORE	Specifies DATA statement to be read by READ statement	141

No	Statement	Function	Page
53	SAVEM	Saves machine language program onto disk	142
54	SCREEN	Declares text window, graphics window	143
55	SET	Sets file mode	144
56	SOUND	Generates sound	145
57	SPRITE	Specifies sprite	148
58	STOP	Interrupt program execution	151
59	VERIFYM	Compares machine language program saved on cassette with program in memory	152
60	VPOKE	Write data into VRAM	153

Functions

No	Function name	Function	Name
1	ABS (x)	Gives absolute value of x	154
2	ACS (x)	Gives inverse cosine of x in radian	154
3	ASC (S)	Converts character S into equivalent ASCII code	155
4	ASN (x)	Gives inverse sine of x in radian	157
5	ATN (x)	Gives inverse tangent of x in radian	157
6	CHRS (x)	Converts ASCII code into character or control code	158
7	COS (x)	Gives cosine of x	159
8	DEG (x)	Converts radian to degree	159
9	DSKF	Used to check free space on disk	160
10	FOF ($=x$)	Used to check whether sequential file is read to the end of file −1: yes, 0: no	161
11	EXP (x)	Gives e to the power of x	162
12	FRE	Used to check free space in user memory area	162
13	HEXS (x)	Converts x into hexadecimal numeric character string	163
14	INKEYS	Gives code of the key being pushed	164
15	INP (x)	Gives input value of input board x	165
16	INPUTS (x, #y)	Set character string x-characters long from sequential file #y	166
17	INT (x)	Gives greatest integer not greater than x	166
18	LEFTS (S, x)	Gives substring x-characters long from the beginning of character string S	167
19	LEN (S)	Gives length of character string S	169
20	LGT (x)	Gives common logarithm of x	169
21	LOC (n)	Gives logical offset in file	169
22	LOF (n)	Gives file size	170
23	LOG (x)	Gives natural logarithm of x	171
24	LTW (x)	Gives logarithm of x to the base 2	171
25	MIDS (S,x,y)	Gives substring starting from x-th character, y-characters long, to the end if y omitted	172
26	PEEK (x)	Gives content of memory at address x	173

No.	Function name	Function	Page
27	PI	Gives ratio of circumference of circle to diameter (3.14159265 36)	174
28	RAD (*x*)	Gives equivalent angle in radian of *x* in degree	175
29	RIGHT$ (S, *x*)	Gives substring from *x* th character counting from right to the end of string S	176
30	RND (*x*)	Generates random number	177
31	SGN (*x*)	Gives numerical sign of *x*	178
32	SIN (*x*)	Gives sine of *x* in radian	179
33	SPC (*x*)	Used in PRINT statement, prints *x* consecutive spaces	180
34	SQR (*x*)	Gives square root of *x*	180
35	STICK (*n*)	Detects input direction for joystick *n*	181
36	STR$ (*x*)	Converts *x* into equivalent character string	182
37	STRIG (*n*)	Detects condition of push-button of joystick *n*	183
38	TAB (*x*)	Used in PRINT statement, sets the start position to be *x*-th column from left	184
39	TAN (*x*)	Gives tangent of *x* in radian	184
40	TIME$	Used to see inner clock	185
41	VAL (S)	Converts numeric character string S into equivalent number	186
42	VPEEK (*x*)	Gives content of VRAM at address *x*	187

SEGA ENTERPRISES LTD.